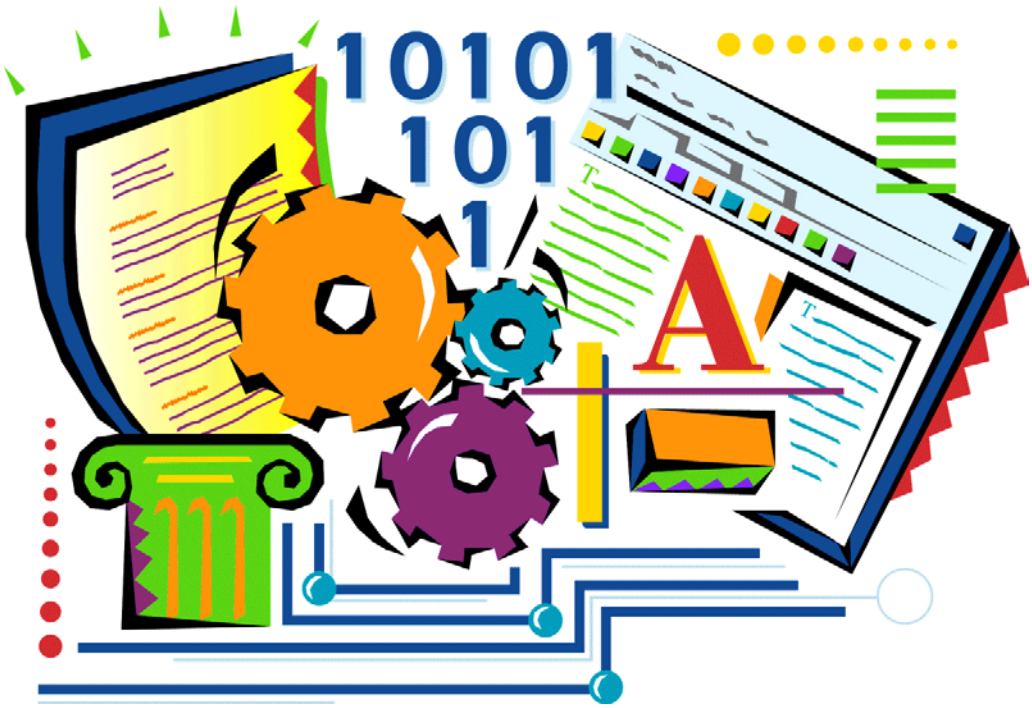
ReportBuilder



Developer'sGuide

Third Edition

ReportBuilder Developer's Guide

Third Edition

CONTENTS

INTRODUCTION

REPORTBUILDER FUNDAMENTALS

MAIN

Introduction

The Delphi Components

Report Components

Smart Layouts

SubReports

Form Emulation

DATA

CODE

Dynamic Configuration

Performing Calculations

Creating Reports in Code

DESIGN

The Report Designer

Dialogs

Toolbars

PRINT

DEPLOY

REPORTBUILDER ENTERPRISE EDITION FUNDAMENTALS

MAIN

Data Dictionary

Putting It All Together

On-Line Help

DATA

Introduction

Query Wizard

Query Designer

Configuring DADE

DADE Architecture

REPORT TUTORIALS

Creating a Report Via the Data Tree

Creating a Report Via the Report Wizard

A Simple Report the Hard Way

Groups, Calculations, and the Summary Band

Using Regions to Logically Group Dynamic Components

Forms Emulation with a WMF Image

Master → Detail Report

Master → Detail → Detail Report

Interactive Previewing with Drill-Down Subreports

Hooking Reports Together with Section-Style Subreports

Using Columns to Create Mailing Labels

Printing to a Text File

Printing from a Text File

Using the JITPipeline to Print from a StringGrid

Using the Rich Text Component for Mail/Merge

Creating a Crosstab

APPLICATION TUTORIALS

Building a Reporting Application

Building an End-User Reporting Application

Adding Data Dictionary Support to the End-User Application

Customizing the Report Explorer Form

Building a Report Application using InterBase

APPENDIX A

Where Do I Go From Here?

INTRODUCTION

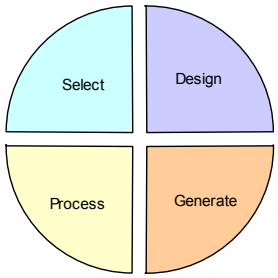
The Basics

Welcome to the ReportBuilder Developer's Guide. This guide is written by ReportBuilder engineers for Delphi developers. The concepts and practical elements of reporting with RB will be revealed throughout this book, and the tutorials will transform the abstract concepts into concrete, useful reports and applications.

Report Creation

Report creation with ReportBuilder can be divided into four main activities:

Report Creation Activities



- **Select** - Refers to the selection of data.
- **Design** - Refers to creating a layout that describes how the document should look.
- **Process** - Refers to the manipulation of the data or the layout in order to control the generation of the document more precisely.
- **Generate** - Refers to the creation of the actual document.

ReportBuilder handles the Generate step for you, and with the help of this guide, you can certainly master the Select, Design, and Process activities. First, let's get a better handle on these rather abstract concepts.

Select

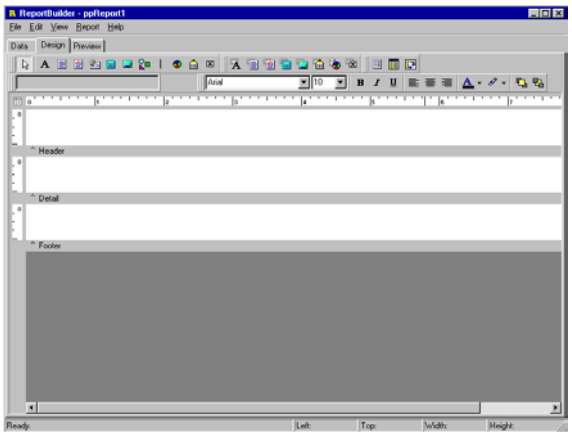
Your data is probably not locked into disposable documents; it's organized (one way or another) as data. When you have data, you generally have a database. Organized data is the first key to creating recyclable documents. ReportBuilder expects data to take a tabular format. Yes, you can create reports based on less structured data, but in general it is most advantageous to have the data organized in a table. Here's an example:

Customer No.	Company	Contact	State
3053	American SCUBA Supply	Lynn Cinciripini	CA
3984	Blue Glass Happiness	Christine Taylor	CA
3054	Catamaran Dive Club	Nicole Dupont	CA
3051	San Pablo Dive Center	Patricia O'Brien	CA
3052	Underwater Sports Co.	Dave Walling	CA

This table contains customer information. In database terms, each row of the table is considered a record. Each column of the table is considered a field. The field names appear in the first row and are not considered part of the data. When you are working on the data selection, the goal is to create a table that will enable ReportBuilder to generate your document correctly. Therefore, you will need to include all the fields you will need for the report, limit the rows selected to only those which should appear in the report, and sort the rows so that they appear in the correct order.

Design

Once data has been selected, you can begin designing your report. You do this by creating a layout. A layout is a combination of objects that describe how the document should look.



This is the Report Designer. As you can see, it looks like many of the other Windows applications you're used to working with. The big difference is that the Report Designer does not contain a document; it contains a layout. The layout can be used to generate many different documents, all based on the data you've selected. The white rectangular areas with the gray bars below are called bands. This report has a Header, Detail, and Footer band. When ReportBuilder generates a document from this layout, the objects in the Header band will appear at the top of each page. The objects in the Footer band will appear at bottom of each page. And the objects in the Detail band will repeat down the page until no more page space is available, at which point a new page will be started. The Detail band prints once for each row of your data selection. This is how a document is created from the layout. You can generate a different document from the same layout by simply changing your data selection.

Process

When you create a layout, you are telling ReportBuilder exactly how you want the document to look. But what if your document is so complex that you can't design a layout to describe it? Though ReportBuilder layouts are quite flexible and powerful, as you will soon see, even the most flexible layout is still fixed. The Process activity of document creation allows you to provide additional instructions to ReportBuilder regarding how the document should be created. Therefore, you can change the report layout as the document is generating. You can also use the processes to format data and perform calculations. Processes are created using ReportBuilder's native language. The language is called RAP (for Report Application Pascal). It is easy to learn and fun to use, especially when you see the cool effects you can create with it. For example, you can use RAP to combine the First and Last Name fields of a contact so the data is easier to read in the report, or to calculate a weighted average to appear in the summary section at the end of a report.

Most reports do not require a process, so don't worry about whether or not you can master this activity. This guide will show you how to do simple calculations and even some fairly complex stuff, if you're interested. Just remember that Process is the place you go when you can't get a report to look exactly the way you want and all other avenues seem to be closed.

Generation

Generation is what happens when you click on the 'Preview' tab in the Report Designer and see the generated document in the print preview window. The document will either look right, or it won't. If it doesn't look right, then it's time to return to the Select, Design, or Process area and make the changes necessary to get the document generating properly. Sometimes you learn the most by tinkering with a data selection or layout and then checking to see how ReportBuilder generates the document differently.

Well, that's ReportBuilder document creation in a nutshell. Now let's jump in and create our first report!

A Quick Test Spin

Overview

This tutorial will show you how to do the following:

- Configure Delphi data access objects for use by reports
- Work with the Report Designer
- Create and configure the most common report components

Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and blank form.

Create a Table, DataSource, and Data Pipeline Component

1 Select the Data Access tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblCustomer
TableName	CUSTOMER.DB

4 Double-click the Active property in the Object Inspector.

5 Add a DataSource component to the form.

6 Configure the DataSource component:

DataSet	tblCustomer
Name	dsCustomer

7 Select the RBuilder tab of the Delphi component palette.

8 Add a DBPipeline component  to the form.

9 Configure the DBPipeline component:

DataSource	dsCustomer
Name	plCustomer

Create a Report and Connect it to the Data

1 Add a Report component  to the form.

2 Configure the Report component:

Data Pipeline	plCustomer
Name	rbCustomerList

Invoke the Report Designer

1 Double-click on the Report component to display the Report Designer.

2 Size and Move the Report Designer window so that the Object Inspector is visible.

Place a Label Component in the Header Band

1 Right-click over the white space of the header band and access the Position... dialog.

2 Set the Height value to 0.9167 and click the OK button. The header band should expand in height.

NOTE: You can also change the height of the header band by dragging the gray divider that appears below the white space of the header band. This method is quick and easy, but not as precise.

3 Click the Label component icon  on the Report Designer component palette.

4 Click on the left side of the header band. A label will be created in the header band and will become the current selection in the Object Inspector.

5 Locate the Edit box at the upper left corner of the Report Designer.

6 Select the text inside it and replace it with Company.

7 Use the font controls to set the font:

Font Name	Times New Roman
Font Size	12
Font Style	Bold & Italic

NOTE: The font controls appear to the right of the Edit box and provide the following functions:

- Font Name drop-down list
- Font Size drop-down list
- Bold
- Italic
- Underline Font Style
- Text Alignment
- Font Color
- Highlight Color

Place a DBText Component in the Detail Band

1 Click the DBText component icon  on the Report Designer component palette.

2 Click in the detail band. A DBText component will be created.

3 Locate the two drop-down lists at the upper left corner of the Report Designer. The list on the left contains a list of data pipelines. It should default to the Customer data pipeline since this is the data pipeline for the report. The list on the right contains a list of fields for the currently selected data pipeline.

4 Select the Company field from the field list. 'Kauai Dive Shoppe' should be displayed in the DBText component.

5 Use the font controls to configure the font:

Font name	Times New Roman
Font size	10
Font color	Black

Preview the Report at Design-Time

1 Click the Preview tab in the Report Designer. You should get a four-page report.

Preview the Report at Run-Time

1 Close the Report Designer.

2 Select the Standard tab of the Delphi component palette.

3 Add a Button component to the form.

4 Configure the Button component:

Name	btnPreview
Caption	Preview

5 Put the following code in the OnClick event of the button:

```
rbCustomerList.Print;
```

6 Run the Project.

7 Click on the Preview button. The report should be displayed in the Print Preview form.

The Best Way to Learn ReportBuilder?

Start simple, then go to the next easy level.

This is the secret of champions. If you work your way from one easy level to the next, then you can end up a very skillful developer. If you jump in at the top, you can end up quite overwhelmed. When you begin to use ReportBuilder, do the easiest report you can imagine. Better yet, complete the 'Quick Test Spin' section of this manual: it will show you how to build a simple report (and it will only take about 5 minutes). I know some of you will read this and say, "But I don't have time for that. I need to know if ReportBuilder will do my real reports." My answer is that you don't have time to not do it. While you burn away hours attempting to complete real reports, you will be learning from experience, which is not a bad exercise in and of itself, but this approach is fairly awful when compared with learning from the master. Give yourself at least 2-3 hours to go through the tutorials. They have been ordered from simple to complex. The first few tutorials are the easiest; the last few are the most difficult. This gradient approach should allow you to build up your ReportBuilder skills with minimum frustration. Regardless of whether you decide to use this manual or not, you should remember this important concept: start simple, then go to the next easy level.

Use the online help.

The help is accessible in three different ways: by accessing the Help | Contents menu option from the Delphi main menu (ReportBuilder Reference should be listed in the table of contents), by selecting a component or property and clicking the F1 key (the appropriate help topic should automatically display), and by searching the help index for a topic. The help contains over 1,500 topics written for the ReportBuilder developer. It's worth using often.

Use this manual.

This manual is designed to give you hands-on experience with report building and conceptual background material that should allow you to better utilize ReportBuilder.

Run the examples; Study the examples; Know the examples.

The examples that come with ReportBuilder are an invaluable part of the product. Not only do they look nice and showcase many of the product's features, they also show you how to build different types of reports. Many of the examples were put together in direct response to questions developers have posed about how to use ReportBuilder. For instance, one example shows you how to compile and use a report as a DLL. Study the examples thoroughly and use them as a reference when you are trying to build a report and not quite sure how to put all the pieces together. If you're having a problem getting something to work, refer to the examples and see whether there is an example of a similar report. The examples are located in the \RBuilder\Demos directory.

NOTE: If you have completed all of the tutorials in this manual, you will be able to pick the examples apart and identify key techniques in short order.

Elements of the User Interface

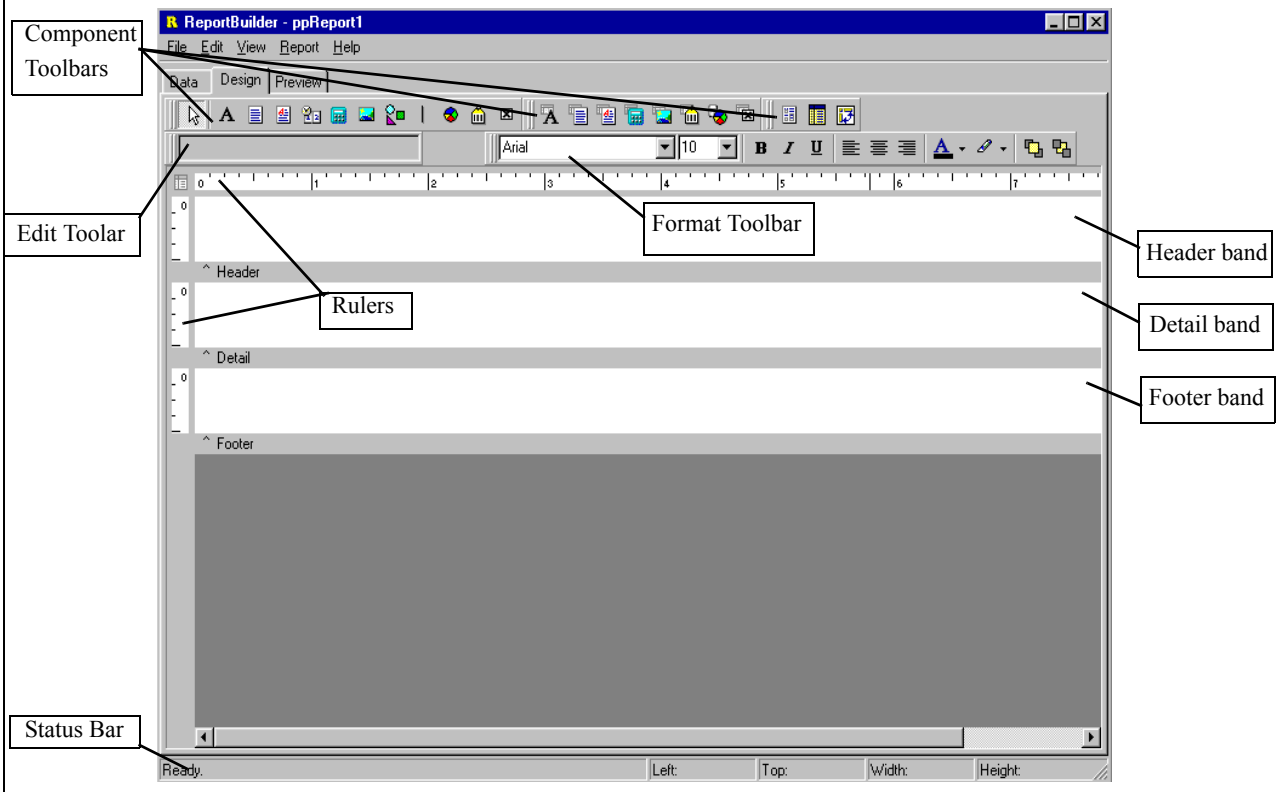
The Report Designer

The Report Designer is your key to productivity when creating reports in Delphi. The ReportBuilder engineers have made every effort to ensure that the Report Designer interface is consistent with other Windows programs you've used before. The major areas of the Designer are listed below.

Component Palette Toolbars

These toolbars are used to create new components. To create a component, click on the icon and then click in the white space of a band. There are three component toolbars: Standard, Data, and Advanced. Use the Standard components to create text, lines, shapes, memos, richtext, etc. Use the Data components when you want to display the data from a database. Use the Advanced components when you need to create more complex reports using subreports, regions, or crosstabs.

The Report Designer



Edit Toolbar

The Edit toolbar allows you to set the most important property for a given component. For example, when a Label component is selected, an edit box appears that allows you to set the Caption. When a DBText component is selected, two drop-down lists appear that allow you to set the Data Pipeline and the DataField.

Format Toolbar

This toolbar appears to the right of the Edit toolbar. It's used to configure the font of textual components and to control component layering via the Bring to Front and Send to Back commands.

Rulers

The horizontal ruler allows you to determine a component's position on the page. The vertical ruler for each band allows you to determine a component's position relative to the starting print position of the band.

All Bands

Notice the gray rectangular area below the white space of each band. This area is draggable, and it allows you to redefine the height of the band.

Status Bar

The Status Bar shows messages and object positions.

Working with the Report Designer

Overview

The Report Designer is actually a sophisticated component editor for the report component (other examples of component editors in Delphi are the Menu Editor of the TMenu component and the Fields Editor of the TTable and TQuery components). The Report Designer enables you to quickly and easily lay out complex reports in much the same way as the Delphi Form Designer.

In Delphi, when you place a component on a form, three things occur:

1 The component is drawn on your form.

2 The declaration of the component is added to your form unit (.pas file).

3 The design-time properties and events of the component become visible in the Object Inspector.

In ReportBuilder, when you place a ReportBuilder component from one of the Report Designer's component palettes on a report band, three things happen:

1 The component is drawn on your report layout.

2 The component is added to your form unit (.pas file).

3 The design-time properties and events of the component become visible in the Object Inspector.

In other words, report components are just like any other Delphi components: you can configure their properties and assign event handlers to them using the Object Inspector. Alternatively, you can use the Report Designer.

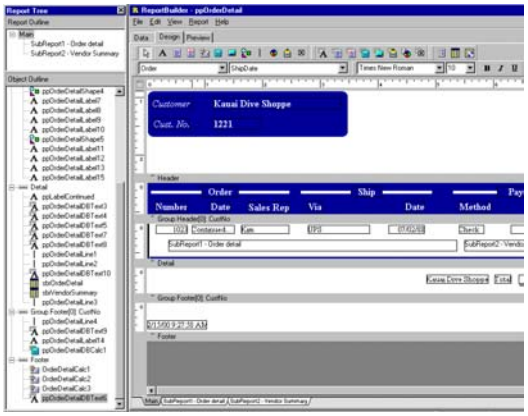
Some tips to help you get the most out of ReportBuilder:

- Maximize the Report Designer window and use the speed menus as much as possible.
- To set display formats, make your dataset active and then use the speed menus to access the Format dialog. The Format dialog will determine the data type of the DataField assigned to the component and display several formats appropriate for that data type. (For example, a field of type date would provide a list of commonly used date formats.)
- To resize the bands using the mouse, position the mouse over the gray rectangular area below the white space of the band, press the left mouse button, and drag.
- To access online help for a component, select the component and Press F1.
- To make a particular band appear as the selected object in the Object Inspector, position your mouse cursor anywhere over the open white space of the band (not over a component in the band) and click the left mouse button.
- To make the report appear as the selected object in the Object Inspector, position your mouse in the top left corner of the Report Designer's work area (to the left of the horizontal ruler and above the vertical ruler) and click the Select Report icon  .
- When working with the Report Designer in its maximized window state, you can bring the Object Inspector to the top by selecting the Report Designer's View | Object Inspector menu item or by pressing F11.

- You can cut, copy, and paste one or more report components at a time either in the same report or between different reports.
- Avoid sharing a datasource for a ReportBuilder report with a data-aware control on your Delphi form (for example a DBGrid). This is inefficient for reporting because when ReportBuilder traverses the dataset to generate a report, your data-aware controls will be repeatedly notified that the current record has changed. If you have any events attached to the data-aware controls, the events will fire repeatedly as well. This slows performance greatly and can cause unexpected results.
- Use the PageLimit property of the report to limit the number of pages previewed when working on long reports. In other words, if the report is 200 pages long, set the PageLimit to 20 and only the first pages will be previewed.
- Close the Report Designer window prior to closing your form or exiting Delphi. Otherwise, an access violation may occur.

The Report Tree

You can display the Report Tree by selecting the View | Toolbars | Report Tree menu option. The report tree shows all of the bands and components in the report. The components for each band are listed in layered order (the order established via the Bring to Front and Send to Back commands). The bottommost component is listed first; the topmost component is listed last.

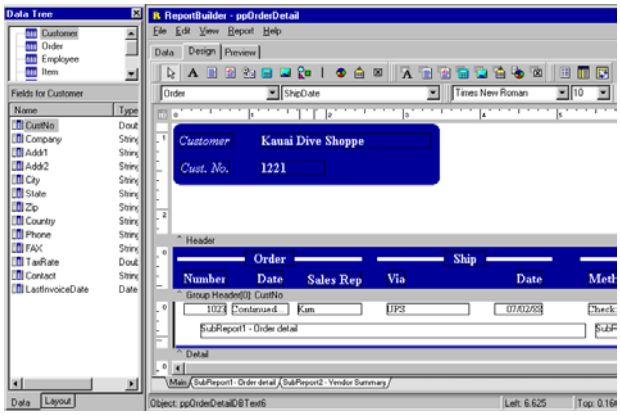


You can use the Report Tree to see exactly what components are contained in the report and to select individual components.

You can turn on the Report Outline by right-clicking over the Report Tree. The Report Outline is useful when you have subreports in your report. You can select any subreport in the Report Outline and the bands and components for that subreport will appear in the Report Tree. (The subreport will also be displayed in the Report Designer.)

The Data Tree

You can display the data pipelines that can be used to create data-aware components within the report by selecting the View | Toolbars | Data Tree menu option. The data tree shows a list of data pipelines in the top window and a list of fields for the currently selected data pipeline in the bottom window.



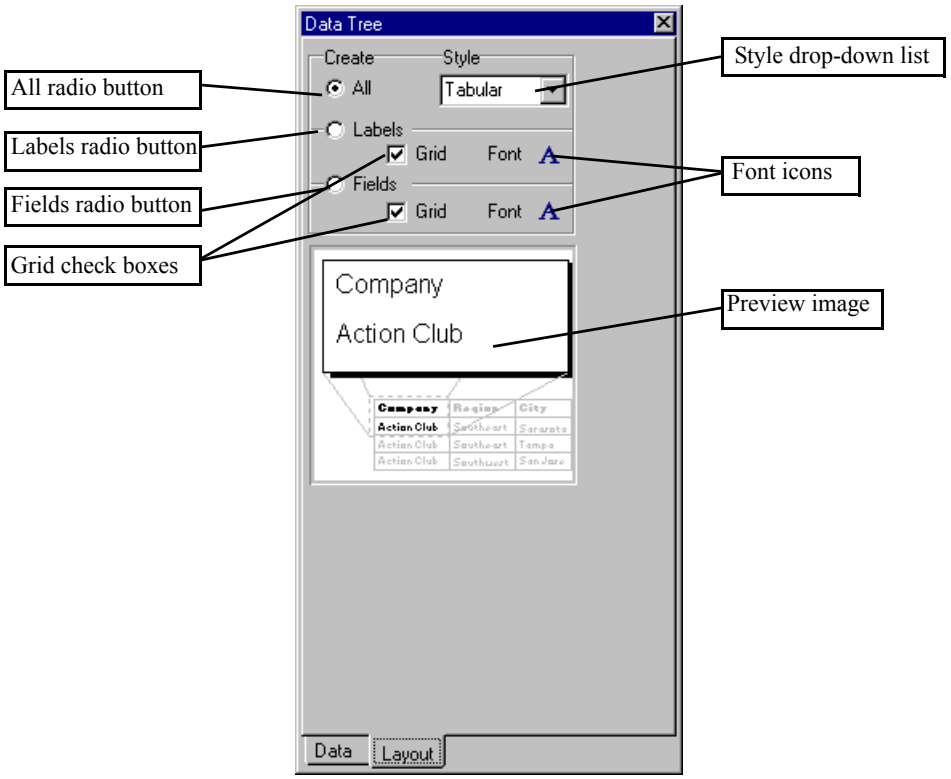
You can select multiple fields in the field list and drag them into any band. Data-aware components and corresponding labels will then be created. Notice the 'Data' and 'Layout' tabs at the bottom of the Data Tree. You can use the Layout tab to customize the behavior of the Data Tree's drag-and-drop capabilities.

Data Tree - cont.

The Layout tab of the Data Tree contains many settings which you can use to customize the drag-and-drop capabilities of the Data Tree.

Style drop-down list Controls whether the DBTextand Label components are oriented in a columnar or stacked fashion .

Data Tree Layout Tab



All radio button
When selected, both DBText components (assigned to the selected fields) and corresponding label components are created.

Fields radio button
When selected, only DBText components (assigned to the selected fields) are created.

Label radio button
When selected, only Label components (with the captions set to the field name) are created.

Grid check boxes
Controls whether a shape is placed behind the DBText or Label component.

Font icons
Controls the font name, size, style, and color via a standard font dialog.

Preview image
Shows how the created components will look.

Reporting Basics

Lookup Tables/Queries

A data-aware report component can be set to any data pipeline on your form; therefore, if a lookup table or query is connected to your master or detail data, you can assign fields from the lookup data simply by creating a data pipeline for the lookup and then assigning that data pipeline to a component.

Filtering Data

Whenever you set a filter on data that is connected to a report, you need to call the Report.Reset method prior to calling Report.Print. This will notify ReportBuilder that it needs to re-access the data and regenerate the report pages, rather than use the internal engine cache. Calling Report.Reset is a good technique whenever it appears that a report is not regenerating in response to changes in the data.

Performing Calculations

There are three ways to perform calculations for reports: Delphi calculated fields, TppDBCalc components, and TppVariable components.

1 Use Delphi calculated fields when you want to calculate a result for each record in the dataset based on the values of one or more fields in the record. You can create TField objects by double-clicking on the dataset component (TTable, TQuery...) and accessing the Fields Editor. Use the Fields Editor to create calculated TField objects. Then, in the OnCalcFields event for the dataset, add the code to calculate the result and assign it to the TField component.

2 Use TppDBCalc components when you need to calculate a SUM, COUNT, MIN, MAX, or AVG for a group or an entire report. The COUNT DBCalcType can also be used in the DetailBand to display the line number for each record in the report.

3 Use the TppVariable component to perform calculations. Set the DataType property to the desired data type. Right-click over the variable and access the Timing... option. Set the timing as appropriate. Add code to the OnCalc event to perform calculations. If calculations are based on the values of other variables, it may be necessary to set the Calc Order of the variables. To set the Calc Order, access the Calc Order dialog by right- clicking over the white space of the band and selecting the Calc Order... menu option. You can then order the variables for calculation. Return the value of the calculation in the Value parameter of the OnCalc event, or assign the result to one of the following properties: AsInteger, AsBoolean, AsString, AsFloat, AsDateTime, AsDate, or AsTime. The value of the Text property will reflect the calculated value.

NOTE: Always remember that Object Pascal event handlers do not execute at design-time; you must compile and run your project to see the results.

Display Formats

You can specify the formats of a DBText, DBCalc, Variable, or SystemVariable component by setting the DisplayFormat property. DisplayFormat differs from the Delphi implementation in the case of string types. In order to format strings, simply type a valid EditMask into the DisplayFormat property. ReportBuilder will then apply the EditMask to the string value.

NOTE: ReportBuilder ignores any display formats you specify within the TField objects of a Delphi dataset.

Dynamic Bands

Set the Band.PrintHeight property to phDynamic when you want the band to use page space on an as-needed basis, shrinking or stretching to accommodate the report components. When the PrintHeight is set to phStatic, the band uses the exact amount of page space specified by the Height property (unless it is not Visible, in which case it uses zero page space).

Stretching Memos and Shapes

Set the Stretch property of a Memo when you want the Height of the memo to automatically stretch to allow the entire contents of the memo to be printed. If you are framing the memo with a Shape, set the Shape.StretchWithParent property to True and the Height of the shape will stretch to accommodate the height of the memo. Finally, use the ShiftWithParent property of the other report components to determine whether the position of the report component should move as the memo stretches.

Controlling Component Visibility

You can use the BeforePrint event of a band to control which components appear when the band prints. To hide all the components in a band, set the Visible property of the band to False. To hide individual components, set the visible of each component to False.

REPORTBUILDER FUNDAMENTALS

Main

MAIN

Introduction

Overview

ReportBuilder is a development environment that can be used to construct reports, report components, and reporting applications. Because reporting encompasses a very wide range of requirements, it is often difficult to put a limit on what should be expected of a reporting tool. The designers of ReportBuilder reduced this broad set of requirements down to the following equation:

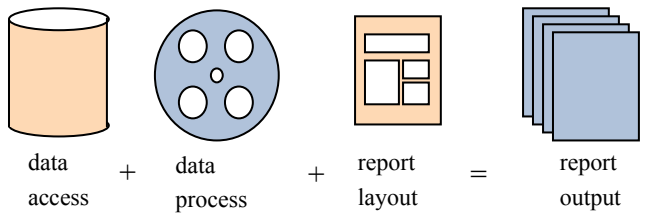
Report Layout

Report layout is a set of components that describe the look and feel of the report and define the behavior of components during report generation.

Report Output

Report output is a set of components that describe the exact content of each page.

The Reporting Equation



Data Access

Data access is the retrieval of data from a database table, text file, Delphi object, or other dataset in an organized fashion (structured as records and fields).

Data Process

Data process refers to the calculation of intermediate results based on data and the modification of the report layout as it generates.

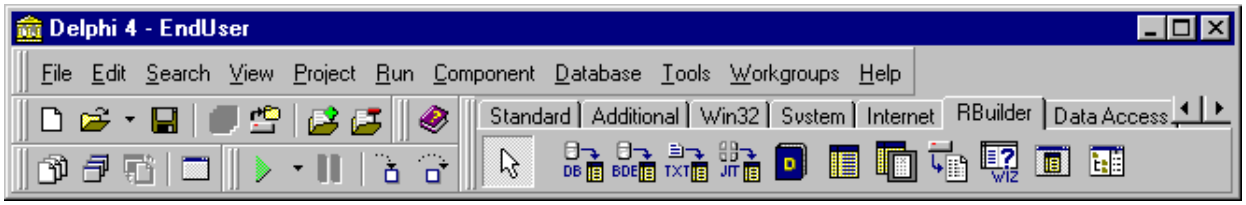
This equation more or less covers the entire area of development known as reporting. In terms of importance, each element of the equation is not supported or is weakly supported, then the utility of the reporting solution is greatly reduced.

Early versions of ReportBuilder focused on the report layout and report output elements, largely due to the fact that Delphi has abundant solutions for the data access (BDE, BDE replacements, FileStreams, StringLists, etc.) and for the data process (events and Object Pascal) elements. In fact, utilizing these solutions is still a valid and productive way for developers to use ReportBuilder. The developer can configure standard Delphi data access components, connect them to a report via the data pipeline component, design and preview the report at Delphi design-time, and, if necessary, code event handlers in Object Pascal to perform calculations or to modify the report layout during generation.

The developers of ReportBuilder discovered that solutions for the data access and data process elements were also needed for end users who were utilizing the ReportBuilder Report Designer as part of a running application. Within the context of a running application, the Delphi developer has no way of providing access to the BDE or to Object Pascal as development environments. End users could not, therefore, avail themselves of these powerful tools. This situation eventually led to the development of DADE (the Data Access Development Environment) and RAP (the Report Application Pascal programming language), which gave end users complete solutions to data access and data process respectively. DADE became available as part of the Professional Edition. Both DADE and RAP are available as part of the Enterprise Edition.

The Delphi Components

Delphi Components



DBPipeline

Used for accessing data via the BDE, third-party BDE replacement products, or TDataSet descendants. The DBPipeline is connected via the DataSource property.

BDEPipeline

In previous versions of ReportBuilder, the BDEPipeline was used for accessing data via the BDE. Though it has been replaced by the DBPipeline, it has been retained for backward capability.

TextPipeline

Used to access comma, tab, and fixed-length record text files. Set the FileName property to specify the file. Double-click on the component to define the field structure.

JITPipeline

Used to access any non-structured data stored in Delphi objects or other sources. Provides total control over the data-access process. Set the InitialIndex and RecordCount properties and code the OnGetFieldValue event to utilize this component. Double-click on the component to define the field structure.

Report

The main component. Double-click to invoke the Report Designer. Assign the DataPipeline property so that the report can traverse data. Assign the DeviceType property to control where the output of the report is directed. Call Report.Print from Object Pascal to print the report or launch the Print Preview Form.

Viewer

This object is rarely used because you can replace ReportBuilder's built-in print preview form with your own customized version very easily (check the Building a Reporting Application tutorial). If you must use this component, an example is provided in \RBuilder\Demos\Reports.

Archive Reader

After you print a report to an archive file (.raf extension), you can read and preview the file via this component. Just assign the ArchiveFileName to the file and call the Print method. In terms of displaying a report, this component works the same as the Report component.

Report Components

Overview

The ReportBuilder Report Component Library (RCL) provides a powerful, robust set of components that have been designed and optimized specifically for the reporting environment. The library includes over 20 components that enable you to put all types of data in your reports: Lines, Shapes,Text, Memos, RichText, Images, Charts, and BarCodes. Advanced components such as Regions, SubReports, and CrossTabs can be used to elegantly model complex reports.

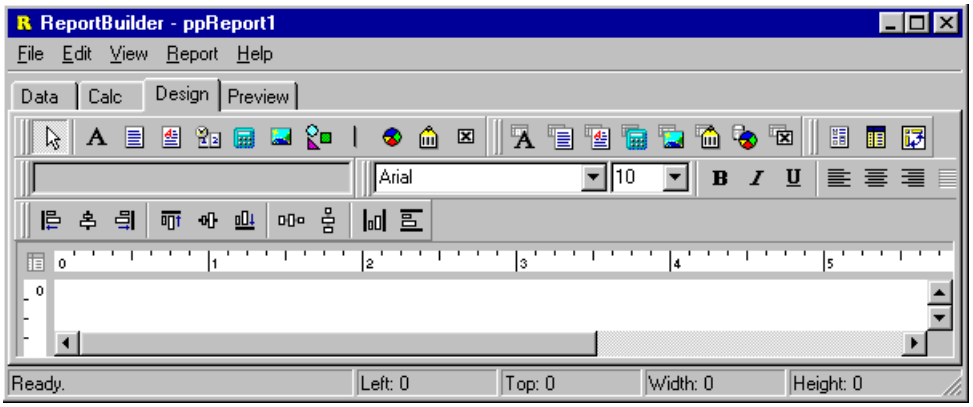
Memo

Used to print multiple lines of plain text in a report. To set the value, assign a string list to the Lines property. To dynamically resize the memo during printing, set the Stretch property to True. Use the ShiftRelativeTo property to define dynamic relationships with other stretchable objects.

RichText

Used to print formatted text. To set the value, assign the RichText property or use the LoadFromFile or LoadFromRTFStream methods. Use the ShiftRelativeTo property to define dynamic relationships with other stretchable objects. At design-time you can use ReportBuilder's built-in RTF Editor to load, modify, and save rich text data stored in files.

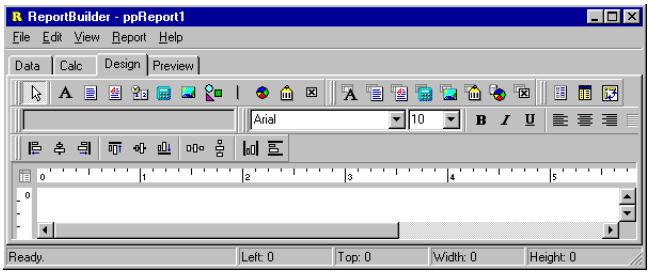
Delphi Report Components



Label

Used to display text. Assign the Caption property to control the text value. To resize the label automatically so it fits a changing caption, set the AutoSize property to True.

Report Components - cont.



SystemVariable

Used to display common report information such as page number, page count, print date and time, date, time, etc. The type of information displayed is controlled by the VarType property. The format is controlled by the DisplayFormat property.

Variable

Used for calculations via an Object Pascal event handler assigned to the OnCalc event or a RAP event handler assigned to the OnCalc event. Access the Calculations dialog (via the speed menu) or the Calc tab of the Report Designer to code a RAP calculation for this component.

Image

Used to display bitmaps and windows metafiles in reports. Assign the Picture property of this component in order to place an image in your report. Use the Report Designer's built-in picture dialog to load images at design-time.

Shape

Use this component to print various shapes (squares, rectangles, circles, ellipses). Set the Shape property to select a type of shape. Use the Brush and Pen properties to control the color and border respectively.

TeeChart

Used to display standard (non-data-aware) TeeCharts. This component enables you to use TeeChart inside the Report Designer. You can access the TeeChart editor via a popup menu.

BarCode

Used to render barcodes. The string value assigned to the Data property is encoded based on the BarCodeType. If the data to be encoded is in a database, use DBBarCode. The following symbologies are supported: Codabar, Code 128, Code 39, EAN-13, EAN-8, FIM A,B,C, Interleaved 2 of 5, PostNet, UPC-A, UPC-E.

CheckBox

Displays a checkbox using the WingDings font.

DBText

Used for displaying values from all types of database fields. Use the DisplayFormat property to format the value.

DBMemo

Used to print plain text from a memo field of a database table. This control will automatically word-wrap the text. Set the Stretch property to True and the component will dynamically resize to print all of the text. Use the ShiftRelativeTo property to define dynamic relationships with other stretchable objects.

Report Components - cont.

DBRichText
Used to print formatted text from a memo or BLOB field of a database table. This control will automatically word-wrap the text. Set the Stretch property to True and the component will dynamically resize to print all of the text. Use the ShiftRelativeTo property to define dynamic relationships with other stretchable objects.

DBCalc
Used for simple database calculations (Sum, Min, Max, Count and Average). The value can be reset when a group breaks using the ResetGroup property.

DBImage
Used to print bitmaps or windows metafiles, which are stored in a database BLOB field.

DBBarCode
Used to render barcodes based on the BarCodeType and the value supplied via the DataField property. The following symbologies are supported: Codabar, Code 128, Code 39, EAN-13, EAN-8, FIM A,B,C, Interleaved 2 of 5, PostNet, UPC-A, UPC-E.

DBTeeChart
Allows data-aware TeeCharts to be placed within a report.

DBCheckBox
Displays a checkbox based on the value of the field specified in the DataField property. Can be used with a Boolean field (or any other type of field via the BooleanTrue, BooleanFalse properties).

Region
Used to logically group components together. Use the ShiftRelativeTo property to move the region in relation to another dynamically resizing component (such as Memo, RichText, or child-type SubReport).

SubReport
Used to handle multiple master details, create side-by-side reporting effects, and hook reports together as one. If you need a report to print within the context of a band, use a child-type subreport. If you need to hook reports together, use a section type subreport. The PrintBehavior property determines the subreport type.

CrossTab
Used to present summarized data in a grid format.

Smart Layouts

Overview
ReportBuilder allows you to create highly dynamic report layouts. The SubReport, Memo, RichText, and Region components have the ability to expand or contract to accommodate the information they contain. There are a host of properties designed to keep your reports looking good in the variety of situations created by these dynamic components.

StretchWithParent
Allows a shape or line to expand or contract based on the change in height of the band or region in which it is contained.

ShiftWithParent
Allows any non-stretching component to move up or down based on the change in height of the band or region in which it is contained.

ShiftRelativeTo
Used to specify the vertical positioning that should take place between multiple stretching components in a band.

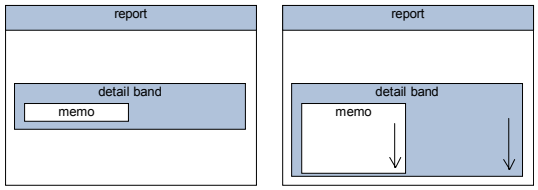
StopPosition (for subreports)
Used to set the position on the page where a child-type subreport will stop printing. Allows a child-type subreport to be confined to a rectangular area of the page.

BottomOffset
Used to create white space between multiple stretching objects that have been linked together using the ShiftRelativeTo property.

OverFlowOffset
Controls the position where a stretching component will begin printing when it overflows to additional pages. This property can be used to print an object at a different starting position when it overflows onto additional pages.

ReprintOnOverFlow
Used to print non-stretching components when stretching components are printing on additional pages.

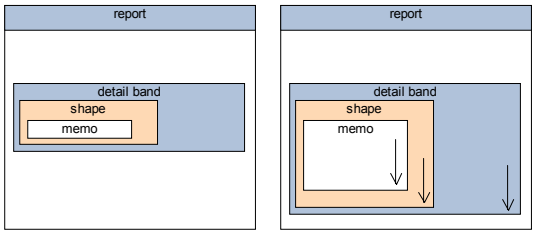
One Memo in the Detail Band



Design Preview

Here we have a single memo component in the detail band. The memo's Stretch property has been set to True. Each time the detail band prints, the height of the memo is recalculated based on the amount of text it contains. As a result, the memo may either grow or shrink in size and the detail band will grow and shrink with it. The memo may contain so much text that it cannot fit on a single page. In this case, the memo will print on additional pages until it is complete. In ReportBuilder, this condition is referred to as overflow. Both the memo and band components have a boolean OverFlow property which can be checked while the report is generating to determine if the memo is printing on an additional page.

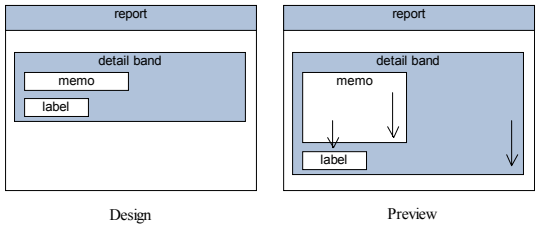
One Memo with a Shape Background



Design Preview

Here we have a single memo in the detail band with a shape behind it. The shape has the StretchWithParent property set to True. The parent, in this case, is the detail band. When the band generates, the memo will stretch based on the text it contains; the band will resize to accommodate the memo, and the shape will resize based on the change in height of the band. This stretching and resizing creates the effect of a border and background for the memo. If the memo overflows onto additional pages, we can also instruct the shape to print by setting the shape's ReprintOnOverFlow property to True.

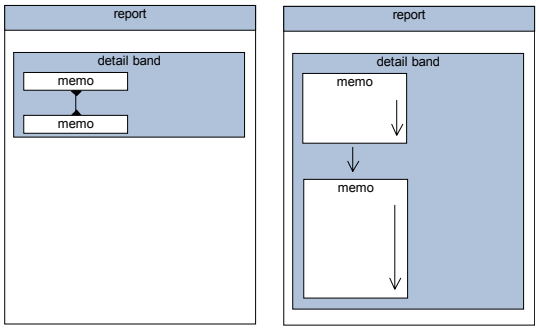
One Memo with Label Beneath



Design Preview

Here we have a single memo in the detail band with a label below it. The label has the ShiftWithParent property set to True. The parent, in this case, is the detail band. When the band generates, the memo will stretch and the band will increase or decrease in height accordingly. The label will shift based on the change in height of the band.

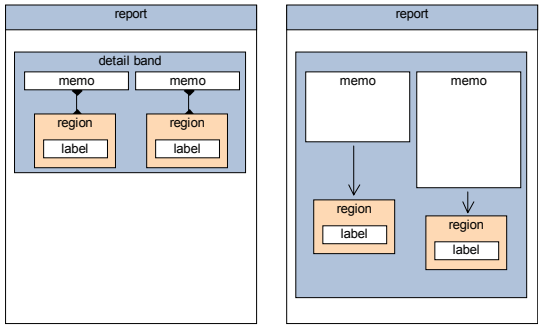
Two Stacked Memos in the Detail Band



Design Preview

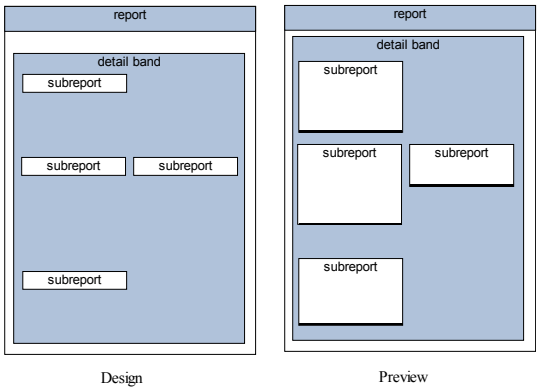
This report requires two memos to be printed, one after the other. This requirement is met by setting the ShiftRelativeTo property of the second memo so that it points at the first. With this configuration, the first memo will print, stretching to accommodate the text it contains, and then the second memo will print to completion.

Two Side-by-Side Memos with Labels Below



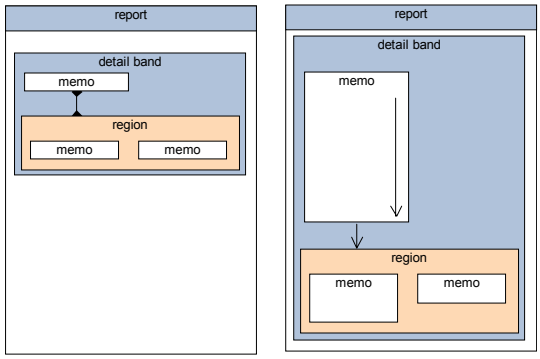
Here we have two memos in the detail band, each set to stretch. When the report generates, the band will grow or shrink to accommodate the memo that contains the most text. Each memo has an associated label below it. We want the labels to shift in relation to the memo above. In order to accomplish this, we place the labels in a region and then set the region's ShiftRelativeTo property to point at the memo above. Now when the report is generated, each label shifts in relation to the associated memo.

Child SubReports in Fixed Positions



This report emulates a form that contains information in fixed rectangular areas of the page. We can get the report to 'fill out' this form by placing child-type subreports at the beginning of each rectangular area and setting the StopPosition property equal to the bottom of the rectangular area.

One Memo with Two Side-by-Side Memos Below



Design Preview

In this report we have a single stretching memo that needs to print to completion, then two additional memos need to print, starting immediately after the first. We can achieve this effect by placing the additional memos in a region and setting the region's ShiftRelativeTo property to the first memo.

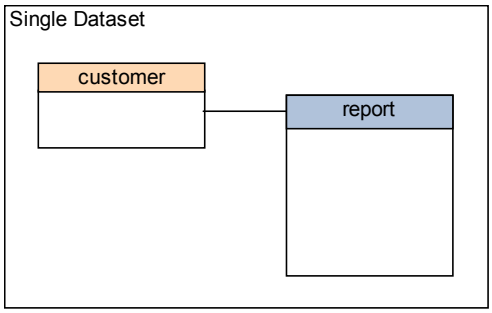
SubReports

Overview

In traditional banded-style report writers, reports that can be printed from a single source of data are quite easy to create. But if the content of the report consists of information from several different sources of data, the choices become quite limited. One option is to use SQL to join the data together into one virtual table, and then build the report based on this table. If many tables are involved, the performance of this approach can be prohibitive.

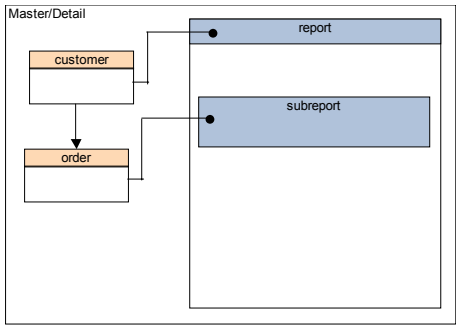
Delphi provides an alternative to this approach by allowing linkages to be established between data access objects. Within ReportBuilder we can use free-form subreports to take advantage of the many configurations these data access objects make possible.

Single Dataset



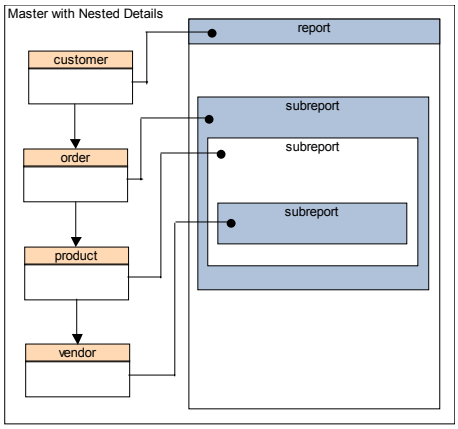
A single dataset can be connected directly to the report via the DataPipeline property. When printed, the report will generate one detail band for each record provided by the dataset.

Master Dataset with Single Detail Dataset



In this scenario, the master data is connected to the detail data via a field or set of fields. It is assumed that this connection results in multiple detail records being selected for each individual master record. The master data is assigned to the report, and the detail data is assigned to a subreport. When the report is generated, the main report will traverse all customer records and the subreport will traverse all orders for each customer.

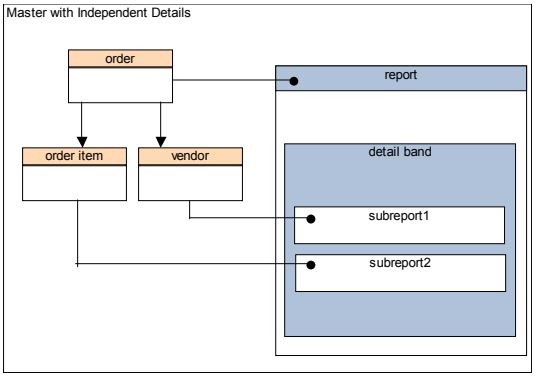
Master Dataset with Nested Detail Datasets



Here we have the master dataset containing a list of customers. Each customer has multiple orders; each order has multiple products; and each product has multiple potential vendors. This configuration is called 'nested' because each set of records is selected based on the linkage established with the previous dataset.

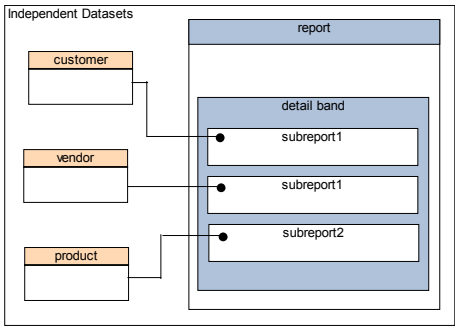
We can traverse this data configuration by nesting subreports in the detail band. The customer dataset is assigned to the main report. The order dataset is assigned to a subreport in the detail band of the main report. The product dataset is assigned to a subreport in the detail band of the product subreport, and so on. In this way, the report is constructed to match the data, and each dataset has a full layout that can be used to render its contents.

Master Dataset with Multiple Independent Datasets



Here we have the order table. Each order has many order items. For each order, there is also a group of vendors that can supply the products for that order. Both of these datasets are linked to the master dataset, as opposed to being nested within one another; therefore, the datasets are 'independent.' This type of data can be handled by placing two subreports in the detail band. The first subreport can be linked to the order item dataset, and the second subreport can be linked to the vendor dataset. In order to print the vendor data after the order item data, we need to link the vendor subreport to the order item subreport via the ShiftRelativeTo property.

Independent Datasets



Here we have three sources of data with no linkage between them. In this report, we want to print all of the customers, then all of the products, and then all of the vendors. The report needs to fit together like a book, with each dataset providing a chapter. Here we use the main report to launch a subreport for each of the datasets. The subreports would be set with a PrintBehavior of section, which means that each subreport would start a new page, generate a set of pages as necessary to traverse all of the data, and then return control to the main report. The main report is not connected to any dataset, and so will print only a single detail band.

Form Emulation

Overview

Form emulation is the process of taking a paper-based or electronic form and rendering a likeness of it. The likeness may include formatting of the form itself, or it may only contain the data that will 'fill-out' the form. There are two basic issues that a form emulation solution must resolve:

1 How will the formatting of the form be generated?

2 How will the data that fills-out the form be generated?

The first issue is resolved by using either a page-sized band within the report or by utilizing a page style. A page style can be designed just like a band, but generate as a background to the bands of the report.

The second issue is resolved by either using simple data-aware components or by using more complex region or subreport components.

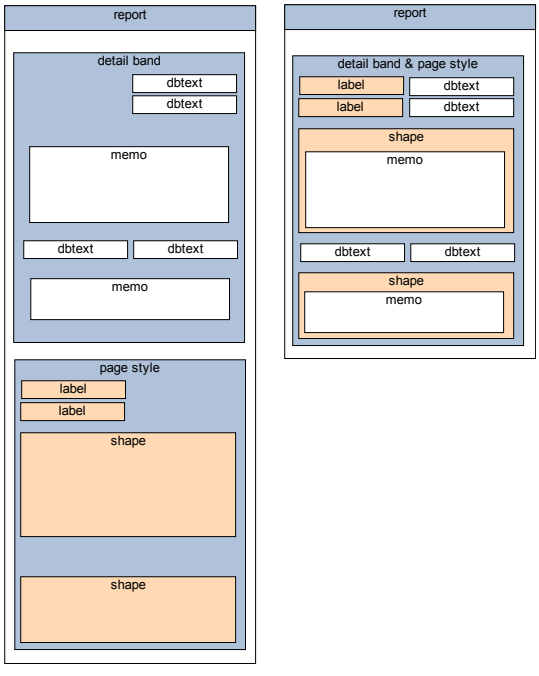
In ReportBuilder, there are several approaches that can achieve form emulation. These approaches are discussed in detail in this section.

Single Page Forms

Single page forms can be emulated in several ways:

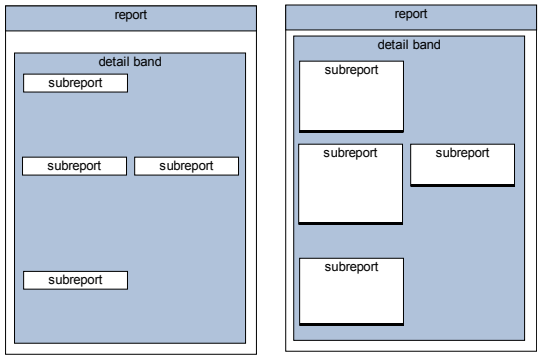
1 Expand the detail band to the printable height of the page, hide the header and footer bands, and place all form formatting and data-aware controls in the detail band. This approach yields one form per record. The following diagram shows an entire form in the detail band.

2 Add a page style to the report. Place all formatting for the form in the page style. Place the data-aware controls in the bands as you would when building a normal report. This approach yields a variable number of records per form, depending on the height of the bands. In this approach, the detail band is used to fill-out the form. The following diagram shows how a detail band can fill-out a page style.



Design Preview

3 Expand the detail band to the printable height of the page. Hide the header and footer band, and place all form formatting in the detail band. Place child-type subreports in different areas of the form where a table or particular source of data is needed to fill-out that part of the form. Place data-aware components in areas of the form where the main data pipeline supplies the data. This placement yields the most flexible and powerful form emulation solution, but is only needed when multiple datasets are used. The following diagram shows subreports filling out a form in the detail band.



The form itself may be too complex to recreate using report components. In this case, it is recommended that you scan the form and convert it to a windows metafile. You can then place the windows metafile in the report as a background for the detail band, or you can place it in the page style, where it will function naturally as a background for the report.

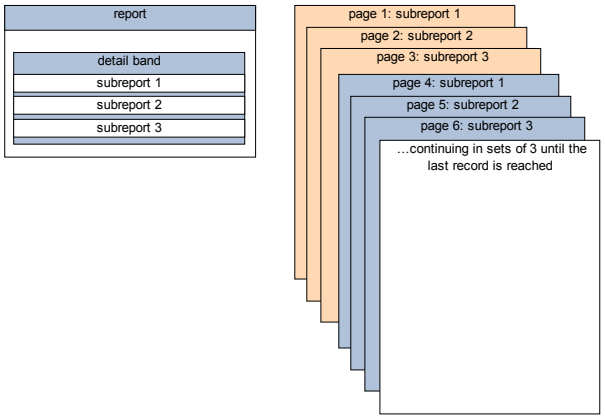
When you have a form in a WMF image, the end user gets an excellent print preview capability. If you are printing on pre-printed forms, you can set the visible property to False when the report is sent to the printer. In this way you can provide a filled-out form in the print preview window, but only the text necessary to fill-out the form is actually sent to the printer.

You can convert a scanned form to a windows metafile image (WMF) via a product such as Transform Suite, by MIPS.

Multi-Page Forms

All of the approaches discussed in the single page form topic are applicable to multi-page forms. The additional problem of organizing single-page forms into sets of multi-page forms can be resolved using section style subreports. When section-style subreports are rendered, they generate a new page within the parent report, continue generating pages until all data has been traversed (or they are stopped manually through a procedure call to the report engine), and then return control to the parent report. One special behavior of section-type subreports is that the parent report generates no pages when the section is placed in a dynamic-height detail band. In this case, the main report is used as a launching pad for sections. Since each section is a full-fledged report in its own right, all of the single page form approaches can apply to each section, thus creating a multi-page form solution.

The diagram below shows three section-type subreports in the detail band of the main report. The main report is assigned to a data pipeline, so the detail band will print once for each record. The subreports contain data-aware controls that point at this data pipeline. This report provides three pages of content for each record, and each page has its own unique format.



DATA

DATA

Introduction

Overview
In ReportBuilder, data access is provided via the data pipeline component. ReportBuilder includes data pipelines for accessing data from a variety of sources.

BDEPipeline
In previous versions, the BDEPipeline was used to access data via the Borland Database Engine (BDE). Though it has been replaced by the DBPipeline, it has been retained for backward capability.

DBPipeline
Used to access data via the BDE, a BDE replacement product, or TDataSet descendant.

TextPipeline
Used to access data in ASCII text files.

JITPipeline
Just-In-Time pipeline for accessing data in Delphi objects.

Regardless of the type of pipeline or the type of data being accessed, the data pipeline component has two basic purposes:

1 To supply data

2 To control data traversal

Supplying Data
Data pipelines provide data via fields. For instance, the following code would retrieve the current field value of a field called 'Company':

```
lValue := DataPipeline1 ['Company'];
```

Each time a data-aware report component prints, it uses this approach to retrieve the data from the data pipeline. Data-aware report components have two properties that determine the data they will retrieve: DataPipeline and DataField. Once these two properties are assigned, the data-aware component has the ability to retrieve data directly from the data pipeline, independent of the report in which the data-aware component resides.

Controlling Data Traversal

The second purpose of the data pipeline is to control data traversal. Data traversal is the act of moving from the first record of the data to the last record. When a report is printed, the report engine traverses the data by completing the following steps:

1 Opens the data pipeline.

2 Goes to the first record.

3 Begins printing the page and then gives the detail band the opportunity to print.

4 Goes to the next record.

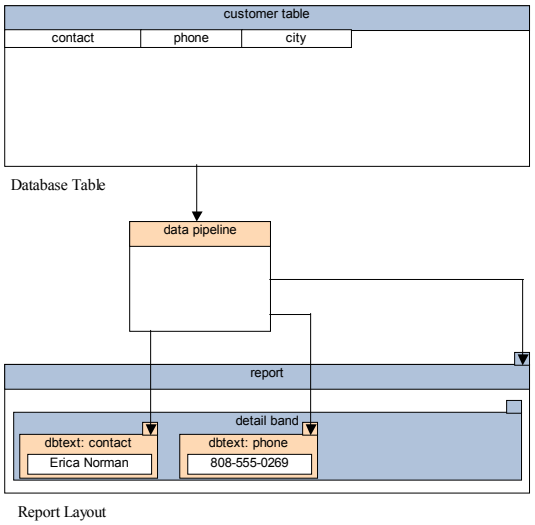
5 Gives the detail band the opportunity to print.

6 Continues steps 4 and 5 until there is no more page space.

7 Completes the page.

8 Continues steps 4 through 7 until all records have been exhausted.

It is important to note that when the detail band is given the opportunity to print, the data-aware components within the detail band are rendered; at this point, they retrieve the field value of the current record. The engine then moves to the next record and prints the detail band again. It is this combination of the report traversing the data and the data-aware components retrieving the data that creates the pages of the report. If the data pipeline is not assigned to either of these entities (data-aware component or report), then the report will not work. Therefore, the report must be assigned to a data pipeline and each data-aware component must be assigned to a data pipeline and a datafield.



We've said that the report engine traverses the data. That isn't completely true. The report engine makes requests of the data pipeline (such as open, first, next, last), and then relies on the data pipeline to do the work. Therefore, the data pipeline controls the data traversal. This control can be used to great advantage.

For instance, let's say you are displaying a database grid on a form. The user has selected an individual record of the grid and wants to print that record. If you have a data pipeline pointed at the same data source as the grid, then it can access

all of the records. However, we can instruct the data pipeline to traverse only the current record by simply setting the RangeBegin and RangeEnd properties to CurrentRecord. When the report prints, it will send traversal requests to the data pipeline, and the data pipeline will traverse only one record. It will then inform the report engine that all records have been traversed and the report will print only one record.

Let's take this example further and say that you let the user select multiple records from the database grid. You then want the report to contain only the selected records. In this case we can assign the Bookmarks from the grid using the AddBookmark method of the data pipeline. When the report prints, the data pipeline will traverse only those records that are in the list of bookmarks and only those records will appear in the report. Essentially, the engine makes the same traversal requests of the data pipeline for every report, but it is the data pipeline that controls how the data is actually traversed.

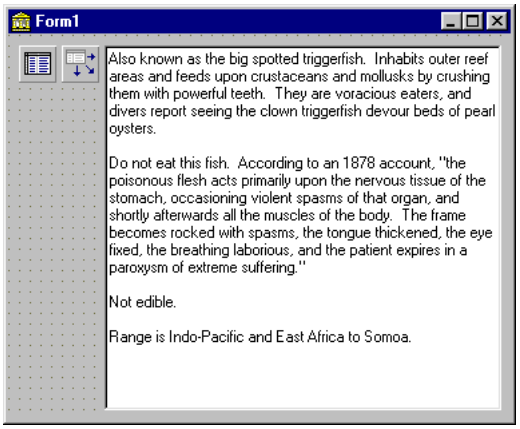
BDE Support

Overview

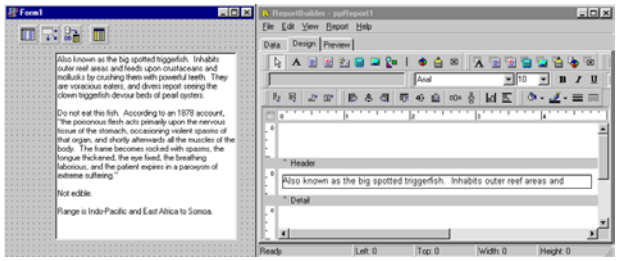
BDE stands for Borland Database Engine. The Delphi development environment utilizes the BDE to access data in desktop database files (such as Paradox or Access) and in client-server databases (such as Oracle and Sybase). One of the reasons why Delphi has been so successful is due to the wide range of data sources that can be accessed via the BDE. ReportBuilder leverages this powerful solution.

Data Access

In Delphi, data access via the BDE can be accomplished using a TTable or TQuery component. A TDatasource component is then connected to the table or query. A data-aware component can then be connected to the datasource. The following screen-shot shows a field value being retrieved into a data-aware memo component.



ReportBuilder augments this data access model by adding another component: the data pipeline. There are several different types of data pipelines, but for purposes of accessing data via the BDE, a DBPipeline component can be used. The following screen shows a DBPipeline retrieving the same field value.



The value is displayed in a ReportBuilder data-aware memo component. You may notice that only the first line of the memo is shown in the ReportBuilder Report Designer. This is due to the fact that when the report generates, the memo component will recalculate the height based on the text it contains. Therefore, there is no reason to specify the height of the memo in the report layout, as it will automatically resize on the generated page.

BDE Alternatives

Overview

The Borland Database Engine (BDE) is what Delphi uses to access desktop and client/server databases. However, there are a number of BDE replacement products available. ReportBuilder's DBPipeline can also be used with these products or any database that supports Delphi's TDataSource. Some of the more popular Delphi database add-on products used by our customers are listed below.

Advantage

by Extended Systems

Provides access to .DBF files. Desktop and high performance Client/Server versions are available. ReportBuilder allows the Advantage TDataSet descendant to plug in directly. The high-performance, low maintenance features of the Advantage Database Server and the extremely well-designed architecture of ReportBuilder make this combination an excellent client/server reporting solution.

Apollo

by Luxent Software

Provides access to .DBF files.

Flash Filer

by Turbo Power Software

Compact database written in Object Pascal.

Opus

by Opus Software GmbH

Provides access to MS Access database files.

Titan

by Reggatta Systems, Inc.

Several products are offered that provide access to BTrieve, MS Access, and SQLAnywhere.

ODBC Express

by DataSoft Pty.

Provides ODBC access without the overhead of the BDE.

ODBC98

by Kosta Corriveau

Provides ODBC access without the overhead of the BDE.

InfoPower

by Woll2Woll Software

Data access components are used to filter, search, etc. Although InfoPower is not a database engine like the products above, the components are widely used by Delphi developers everywhere.

Text Files

The TextPipeline Component

Reports can be printed directly from text files without using a database product. This functionality is provided via the TextPipeline component.

The following formats are supported:

- Comma-delimited
- Tab-delimited
- Fixed-Length records
- Custom-delimited (where you specify the delimiter)

The TextPipeline component is essentially a simple data retrieval engine that enables you to access data in text files in the same manner as data stored in a database table. You can even define master/detail relationships between data in two text files. The TextPipeline contains a Field Editor that is used to define the data fields for the text file. Once the data fields have been defined, you can access the Report Designer and assign those fields to ReportBuilder's data-aware components.

The Field Editor

Each of the data pipelines in ReportBuilder has a Field Editor that is accessible by double-clicking on the pipeline component at design-time. The TextPipeline's Field Editor is pictured below. Notice that the View Data button has been activated to display the contents of the text file. This feature can be a useful aid when defining the fields for the text file.

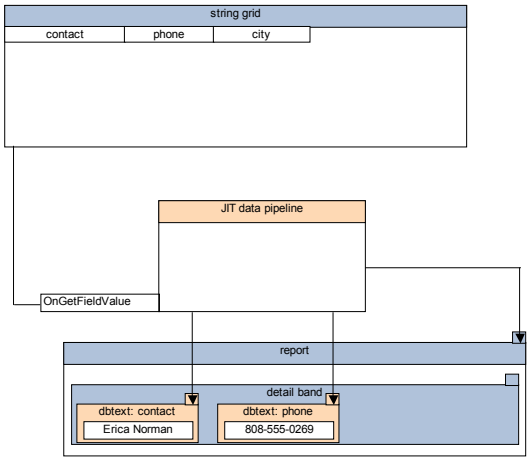
Delphi Objects

Overview

Data pipelines present a structured set of data to a report. This structure takes the form of records and fields. The report engine expects operations like open, first, next, and last to provide a certain response from the data pipeline. The data-aware components within the report expect field values to be retrievable by simply passing a field name to the data pipeline. In the DBPipeline component, this functionality is implemented by calling the methods of a TDataSet object. However, this is not the only way a data pipeline can provide the necessary data access functionality to a report. The JITPipeline (JIT stands for Just-In-Time) triggers events to accomplish the same results as the DBPipeline.

JITPipeline

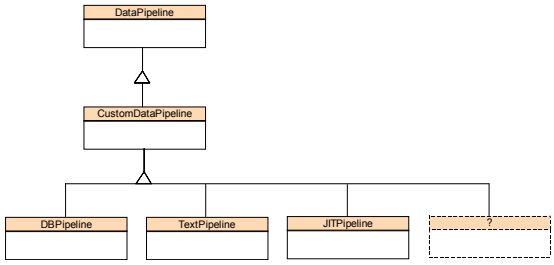
The JITPipeline provides a set of events that, when coded properly, allow reports to print from any source of data referenced by the event handlers. The following diagram shows a JITPipeline that is assigned to event handlers referencing a standard Delphi string grid. In the report, data-aware components are created just as they would be for any other database report - by assigning field names from the JITPipeline to the data-aware controls in the report. This results in a clean, maintainable implementation. If you later decide to provide data to the report via a text file or database table, you can simply swap out the JITPipeline with a new pipeline of the correct type.



Native Access to Proprietary Data

Overview

If you have a considerable amount of data in a proprietary format, and many reports need to be created based on this data, the highest level of maintainability and ease of use will be provided by a custom data pipeline component. This type of component can be created by descending from the TppCustomDataPipeline class and implementing the necessary methods. The end result of such an effort will be a new data pipeline component that can be installed into the Delphi IDE and used on the same basis as the other data pipelines that are provided with ReportBuilder. The object model for the ReportBuilder DataPipeline classes is shown below:



CODE

CODE

The Delphi Event Model

Overview

In the context of Delphi components, events have two fundamental qualities:

1 Events fire as the logical result of an action taken on or by a component.

2 Events fire at a moment when a meaningful action can be taken.

Significance

An event is something that happens as the logical result of an action taken on or by a component. For instance, an action taken on a button control is a mouse-click. In response, a TButton control will fire its OnClick event. An example of an action taken by a component would be the Next method of a TTable. When this method is called, the OnCalcFields event (among others) fires.

Timing

Events fire at a moment when a meaningful action can be taken. In the case of a TButton.OnClick event, the meaningful action would be the one intended by the developer as the basic purpose of the button. This purpose might be to launch another window or close the current form. But not every action is appropriate within the context of the event. It would certainly not be appropriate to free the button in the OnClick or to call the OnClick event handler from within itself. These calls might be OK for other events of the application, but not within the context of the OnClick event.

Essentially, events have both a timing aspect and a significance aspect. Both of these aspects must be taken into account when creating an event handler. The following table lists some of the most important events of the ReportBuilder report component.

This list of events is not exhaustive, but it does provide guidelines for the use of the events. Event handlers can do just about anything, but understanding the intended use of an event can definitely help in selecting the right event. Every component within ReportBuilder has a rich set of events. The exact timing and purpose of these events are provided in the on-line help.

AfterPrint

This event is instantiated after the engine has completed generating the report, but before any supporting forms have been closed. It allows print statistics to be incremented or datasets to be closed.

BeforePrint

BeforePrint is instantiated after all supporting forms have been created and displayed, but before the engine begins generation of the report. It allows the report layout to be modified.

OnCancel

When printing to the printer, OnCancel fires when the Cancel button of the cancel dialog is clicked. It allows cancelled print jobs to be tracked.

OnPreviewFormCreate

This event is instantiated when the Print Preview form has been created, but before it has been displayed. It allows the Print Preview form to be configured.

Dynamic Configuration

Configure Reports During Generation

When we talk about configuring a report while it is generating, we mean anything from setting the caption of a label to conditionally controlling the visibility of an entire subreport. Always keep in mind that ReportBuilder reports are comprised of a collection of objects: Report.Bands[].Objects[]. The report object is the parent of a collection of band objects. Each band object represents a rectangular area of the page and contains a collection of printable objects such as text, images, and memos. All of these objects have properties and events. The events fire while the report is generating, thus enabling us to manipulate the properties and control the behavior of the report.

In order to get a hands-on feel for how this really works, let's code some event handlers that will give you an idea of what can be done while a report is generating.

Font Color

The event handler below is assigned to the OnPrint event of a DBText component. When the component prints, the current value of the PRICE_CHG field is checked. If this value is less than zero, the component prints in red; otherwise, the component prints in black.

It is important to note that the font color is set to the appropriate color every time the component prints. If the code below simply set the font color to red when the value was negative, then the first negative value would print in red and all subsequent values would print in red, regardless of the sign of the value. When setting report component properties, remember to handle all cases, as components remain in any state to which they are set.

Code *Setting the color of a numeric DBText component based on the sign*

```
procedure Form1.ppDBText4Print(Sender: TObject);
begin

   if(Table1.FieldByName('PRICE_CHG' ).AsInteger >= 0)
   then
     ppDBText4.Font.Color := clBlack
   else
     ppDBText4.Font.Color := clRed;
end;
```

Concatenation

This event handler is assigned to the OnPrint event of a Label component. When the component prints, the current value of the FirstName and LastName fields are retrieved. If the FirstName has a length greater than zero, it is included in the caption. The local variables used here are prefixed with 'ls', which stands for local string.

Code *Using a label to display concatenated field values*

```
procedure TForm1.ppLabel3Print(Sender: TObject);
var
   lsFirstName: String;
   lsLastName: String;

begin
   lsFirstName := Table1.FieldByName('FirstName').AsString;
   lsLastName  := Table1.FieldByName('LastName').AsString;

   if (Length(lsFirstName) > 0) then
     ppLabel3.Caption := lsFirstName + ' ' + lsLastName
   else
     ppLabel3.Caption := lsLastName;
end;
```

Address Squeeze

The BeforePrint event of a band is an acceptable place to configure any component within the band. Placing several different component configuration steps in a band-level event handler can make for more maintainable code (as opposed to placing the code in the OnPrint of each component). This event handler retrieves the appropriate field values from a database table, concatenates them in a local string variable, and then adds this variable as a line to the memo.

Code *Using a memo to create dynamically sized address information*

```
procedure TForm1.ppReport1DetailBand1BeforePrint(Sender: TObject);
var
   lsLine: String;
   lsState: String;
   lsZIP: String;

begin
   {clear memo}
   ppMemoAddress.Lines.Clear;

   {add contact}
   lsLine := tblCustomer.FieldByName('Contact').AsString;

   ppMemoAddress.Lines.Add(lsLine);

   {add company}
   lsLine := tblCustomer.FieldByName('Company').AsString;

   ppMemoAddress.Lines.Add(lsLine);

   {add address line1}
   lsLine := tblCustomer.FieldByName('Addr1').AsString;

   if lsLine <> '' then
     ppMemoAddress.Lines.Add(lsLine);

   {add address line2}
   lsLine := tblCustomer.FieldByName('Addr2').AsString;

   if lsLine <> '' then
     ppMemoAddress.Lines.Add(lsLine);

   {add city, state zip}
   lsLine  := tblCustomer.FieldByName('City').AsString;
   lsState := tblCustomer.FieldByName('State').AsString;

   if lsState <> '' then
      lsLine := lsLine + ', ' + lsState;

   lsZIP := tblCustomer.FieldByName('ZIP').AsString;

   if lsZIP <> '' then
     lsLine := lsLine + ' ' + lsZIP;

   ppMemoAddress.Lines.Add(lsLine);

   {add country}
   lsLine := tblCustomer.FieldByName('Country').AsString;

   ppMemoAddress.Lines.Add(lsLine);

end;
```

Continued Group

This event handler is attached to the OnPrint event of a label in the detail band of the report. The 'if' statement checks to see if this is the first page of the group and the first detail band of the page. If it is not the first page of the group, the group is printing on additional pages. In this case, the label is made visible. The label's caption is set to 'Continued...'; this creates the effect of labeling detail lines that print on additional pages for the same group.

Code *Placing a 'Continued...' label in a detail band*

```
procedure TForm1.ppLabelContinuedPrint(Sender: TObject);
begin
   if not(ppReport1.Groups[0].FirstPage) and

      (ppReport1DetailBand1.Count = 1) then
      ppLabelContinued.Visible := True
else
   ppLabelContinued.Visible := False;

end;
```

Regions

The group header band in this report contains two regions. One region contains a detailed set of data; the other region contains a more summarized set of the same data. On the first page of the group, the detailed set of data (in region 1) is displayed. If the group continues onto additional pages, the more summarized version of the data (in region 2) is displayed. The FirstPage property of the group is used to determine if the report is on the first page or additional pages of the group. In the report layout, Region 2 has been placed to the right of Region 1 in order to make the report layout easier to maintain. Therefore, the first time Region 2 is displayed, it must be moved into the same position as Region 1.

Code *Controlling components using regions*

```
procedure TForm1.ppGroupHeaderBand1BeforePrint(Sender: TObject);
begin
   if ppGroup1.FirstPage then
     begin
       ppRegion1.Visible := True;
       ppRegion2.Visible := False;
     end
   else
     begin
       ppRegion1.Visible := False;
       ppRegion2.Visible := True;

       if (ppRegion2.Left <> ppRegion1.Left) then
         ppRegion2.Left := ppRegion1.Left;
     end;
end;
```

SubReports

This event handler is associated with a master/ detail report. The report contains a subreport that prints the detail. This report resides on a form with a button captioned 'Hide Detail'. When the button is clicked, the event handler toggles the value of the subreport's visible property and then prints the report. As a result, the report is either a high-level summary or a full-detail listing; thus, the report provides the functionality of two reports from only one layout.

Code *Controlling the visibility of a subreport*

```
procedure TfrmShowHideDetail.btnDetailClick(Sender: TObject);
begin
   ppSubReport1.Visible := not(ppSubReport1.Visible);

   if (ppSubReport1.Visible) then
     btnDetail.Caption := 'Hide Detail'
   else
     btnDetail.Caption := 'Show Detail';

   ppReport1.Print;

end;
```

Performing Calculations

Overview

Calculations are a vital part of reporting, and ReportBuilder provides a rich set of components and events that allow you to perform a wide range of calculations. Simple calculations can be performed without any coding via the DBCalc component. This component provides Sum, Average, Minimum, Maximum, and Count functions. The calculations can be group-based or report-based. More complex calculations can be achieved via the Variable component. At a minimum, the Variable component requires an OnCalc event handler to be assigned. The timing of this event can be correlated to any number of occurrences within the report generation process. For example, the component can calculate once for each of the following occurrences: the start of the report, a record traversal, a group break, the start of a page, and the start of a column. The Variable component also has an OnReset event that can be similarly correlated. This provides the utmost in calculation flexibility.

DBCalc component

The DBCalc component:

- Performs simple calculations without any coding
- Provides Sum, Average, Minimum, Maximum and Count functions
- Performs calculations that can be group-based or report-based

Variable component

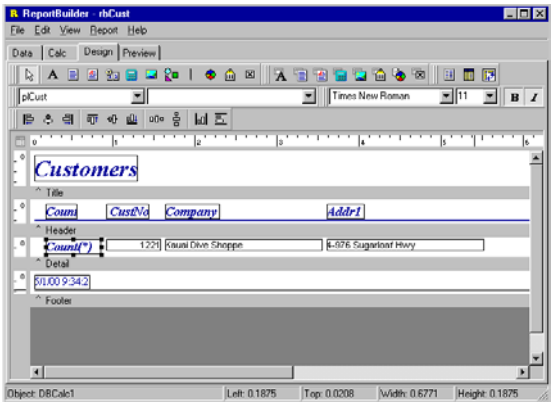
The variable component:

- Requires an OnCalc event handler to be assigned
- Contains properties to control the timing of the OnCalc and OnReset events
- Contains a CalcOrder property to control the ordering of calculations

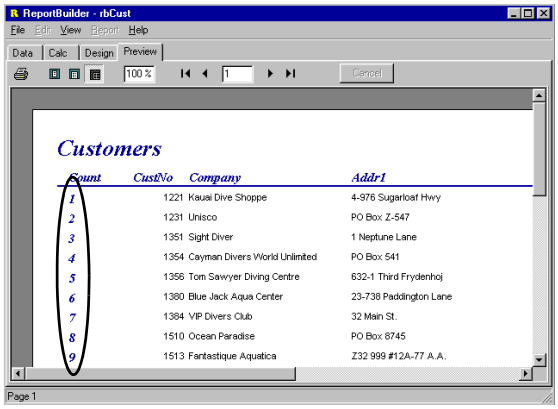
Count

To number the detail bands, place a DBCalc component in the detail band of a report. Right-click over the component and select the Calculations... menu item. A dialog will be displayed. Set the calculation type to Count and click OK. When you return to the Report Designer, the DBCalc will contain this caption: 'Count(*)'. There is no need to select an individual field for the DBCalc because the Count function does not require one. Preview the report. Each detail band will be numbered as shown below.

Using a DBCalc component to number the detail band



Design View



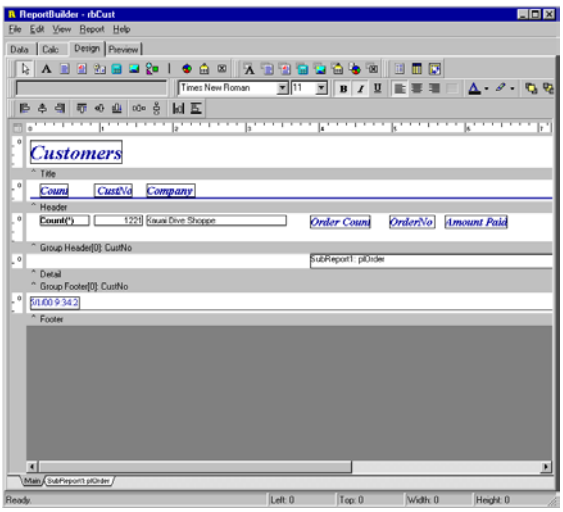
Preview

Count Master/Detail

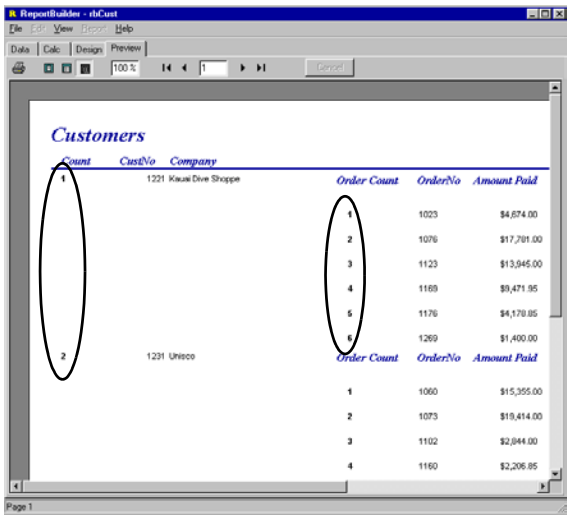
A master/detail report usually contains a group based on the key field in the master table. This allows the fields from the master table to be printed in the group header, as opposed to repeating in the detail band. In order to count the master records, place a DBCalc in the group header band and set the CalcType to count. Then access the Calculations... dialog and clear the Reset Group setting. The ResetGroup is automatically assigned when a DBCalc is placed in a group band. When set, this property causes the DBCalc to reset to zero each time the group breaks. Here we want to count each group (not reset to zero when each group breaks) and so we clear this assignment.

Next, place a DBCalc component in the subreport's detail band and set the Calc Type to count. When the report is previewed, the master records and the detail records will be numbered as shown below

Using a DBCalc component to number the master and detail records



Design View

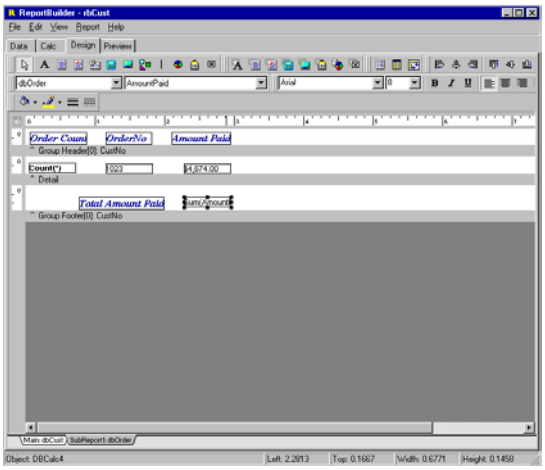


Preview

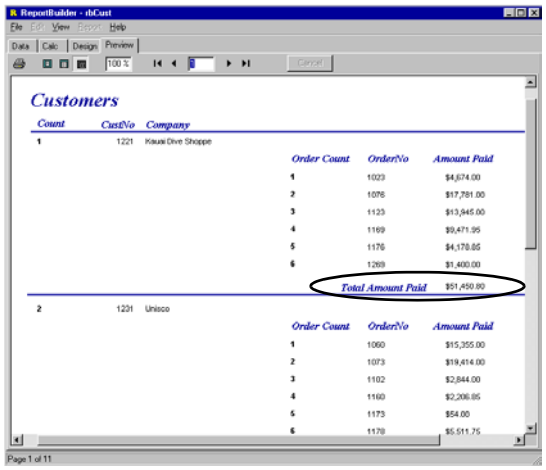
Group Total

Report group totals can be easily calculated by placing a DBCalc in the group footer band. The ResetGroup is automatically assigned when a DBCalc is placed in a group band. Thus, each time the group breaks, the group total will reset to 0.

Using a DBCalc component to number the master and detail records



Design View

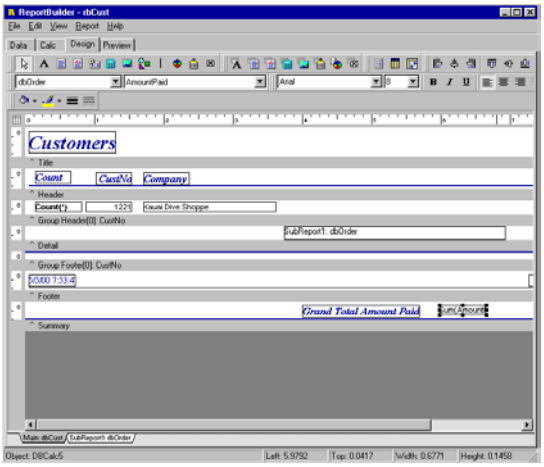


Preview

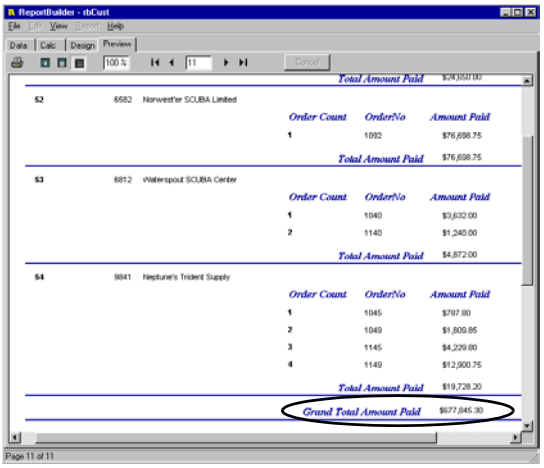
Grand Total

To calculate a grand total, add a summary band to a report and place a DBCalc in the summary band.

Using a DBCalc component to calculate a grand total



Design View

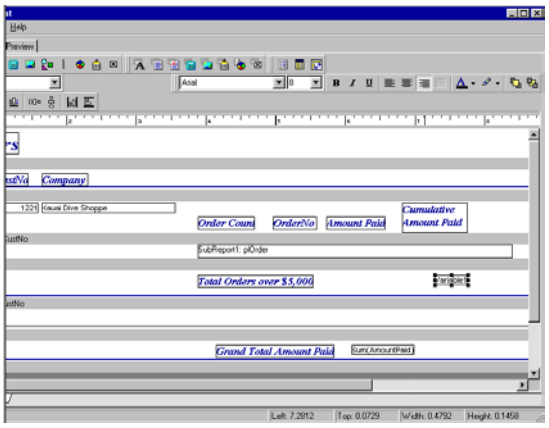


Preview

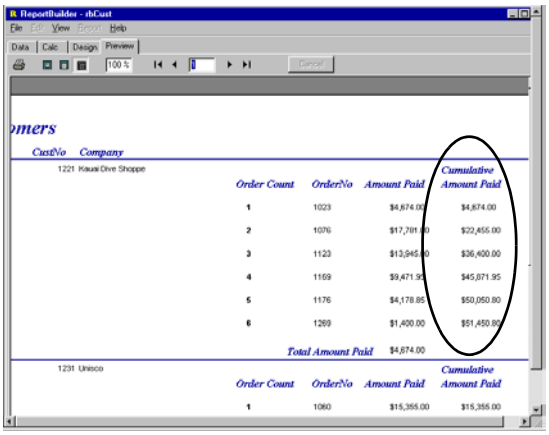
Cumulative Sum

To calculate a cumulative sum in the detail band, add a DBCalc in the detail band and assign it to the appropriate data pipeline and field.

Using a DBCalc component to calculate a cumulative sum in the detail band



Design View



Preview

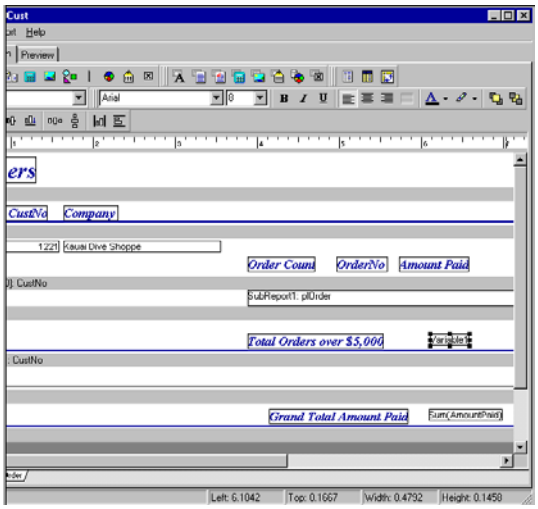
Conditional Group Total

When you need to exclude certain values from a group total, add a Variable component to the group footer band. Access the Timing dialog and set the 'Calculate On' to DataPipeline Traversal, select the appropriate Data Pipeline, then set 'Reset On' to Group Start and select the appropriate group. Code the OnCalc event handler as:

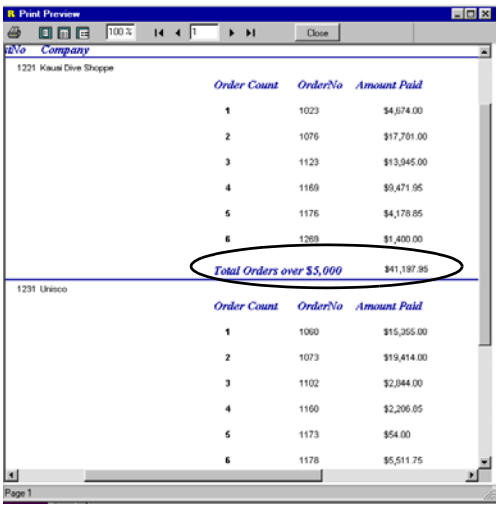
This event handler will accumulate the value of the variable only when the amount paid is greater than $5,000.

```
procedure TForm1.ppVariable1Calc(Sender: TObject; var Value: Variant);
var
   lcValue: Currency;

begin
   lcValue := Table2.FieldByName('AmountPaid').AsCurrency;

   if (lcValue >= 5000) then
     Value := Value + lcValue;
end;
```

Using a variable to exclude certain values from a group total



Design View



Preview

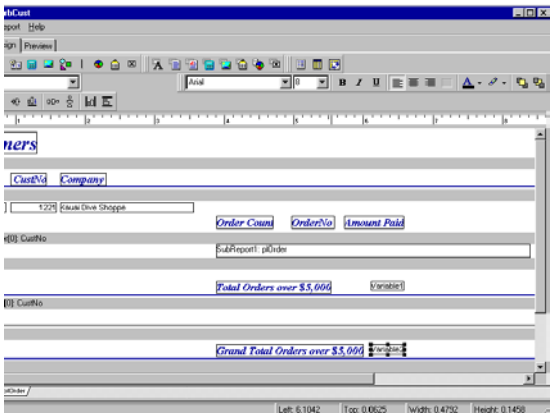
Conditional Grand Total

To exclude certain values from a grand total, add a Variable component to the summary band. Then code the OnCalc event handler as:

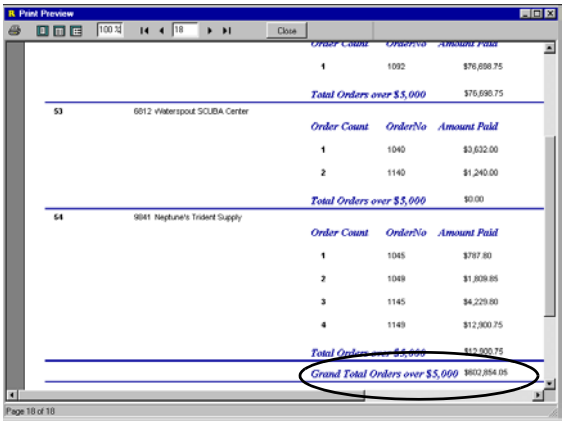
```
Procedure TForm1.ppVariable2Calc(Sender: TObject; var Value: Variant);
var
   lcValue: Currency;

begin
   lcValue := Table2.FieldByName('AmountPaid').AsCurrency;

   if (lcValue >= 5000) then
     Value := Value + lcValue;
end;
```

This event handler will accumulate the value of the variable only when the amount paid is greater than $5,000. The screen shot below shows the result. Notice that this report also has a conditional group total in the group footer band.

Using a variable to exclude certain values from a grand total



Design View



Preview

Look Ahead Total

ReportBuilder has the flexibility to operate as a one pass or two pass report engine. This example shows how to calculate group totals so that they can be displayed in the group header.

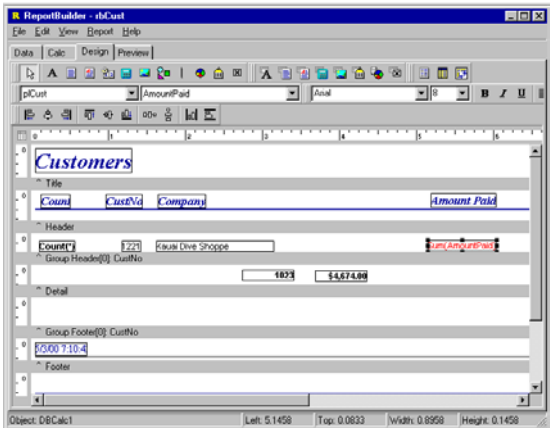
1 Create a group.

2 Place a DBCalc component in the group header band.

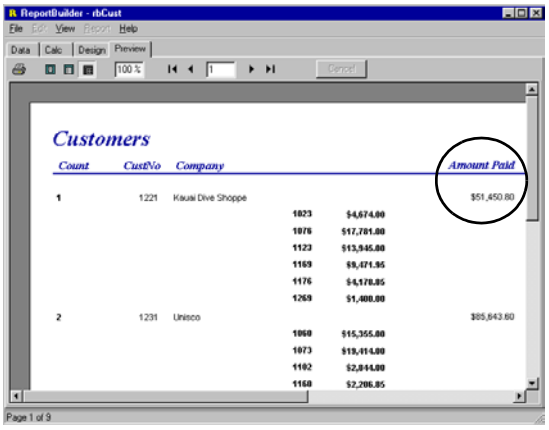
3 Right-click and set the LookAhead property to True.

4 Preview. The total prints before the detail.

Using a DBCalc and a variable to display a group total at the beginning of a group



Design View



Preview

Creating Reports in Code

Coding a Report

A report layout is composed of a set of components. Like other standard Delphi components, the components that make up a report layout have a run-time interface. That is, they can be created and configured using Object Pascal code. This capability can be used to create entire reports dynamically. Take these steps to create a report dynamically:

1 Create data access components.

2 Create the report.

3 Create the report bands.

4 Add data-aware components.

5 Add formatting components.

These are the same steps you would follow to create a report using the Delphi IDE and the ReportBuilder Report Designer. Creating the components manually requires some additional knowledge of how the different parts of the report fit together. The following discussion steps you through the actual code necessary to build a simple report.

1 Declare the necessary uses clause.

```
uses

DB,          {contains TDataSource}
DBTables,    {contains TTable}
ppReport,    {contains TppReport}
ppDBBDE,     {contains TppBDEPipeline}
ppBands,     {contains all band classes}
ppCtrls,     {contains standard components}
ppTypes,     {contains all ReportBuilder
             enumerated types}
ppVar;       {contains SystemVariable and
             Variable classes}
```

The types that are being used from each unit are documented in the comments of this code.

2 Declare the local variables necessary to create the report.

```
procedure TForm1.Button1Click(Sender:
↳ TObject);
var
lTable: TTable;
lDataSource: TDataSource;
lDataPipeline: TppBDEPipeline;
lReport: TppReport;
lLabel1: TppLabel;
lLabel2: TppLabel;
lDBText1: TppDBText;
lDBText2: TppDBText;
lSysVar: TppSystemVariable;
```

The prefix 'l' stands for local. This coding standard makes local variables easily distinguishable from component level variables (which are prefixed with an 'F') and from proper component names (such as 'Report1').

3 Create data access components.

```
lTable := TTable.Create(Self);
lTable.Name := 'tblCustomer';
lTable.DatabaseName := 'DBDemos';
lTable.TableName := 'customer.db';
lDataSource := TDataSource.Create(Self);
lDataSource.Name := 'dsCustomer';
lDataSource.DataSet := lTable;
lDataPipeline := TppBDEPipeline.Create.
↳ (Self);
lDataPipeline.Name := 'plCustomer';
lDataPipeline.DataSource := lDataSource;
```

These are standard Delphi data access components. It is not necessary to assign the Name property of these components; we've done this to give you an idea of how they would be named if they were created within the Delphi IDE.

4 Create the report.

```
lReport := TppReport.Create(Self);
lReport.DataPipeline := lDataPipeline;
```

The data pipeline assignment is vital here. Without a data pipeline assigned, this report would generate an endless number of pages.

5 Create the report bands.

```
lReport.CreateDefaultBands;
```

This method creates a header band, a detail band, and a footer band. It is also possible to create the bands individually and assign them to the report. A report must always have a detail band in order to generate properly.

6 Add labels to the header band.

```
lLabel1 := TppLabel.Create(Self);
lLabel1.Band := lReport.HeaderBand;
lLabel1.spLeft := 2;
lLabel1.spTop := 2;
lLabel1.Caption := 'Customer No.';
lLabel2 := TppLabel.Create(Self);
lLabel2.Band := lReport.HeaderBand;
lLabel2.spLeft := lLabel1.spLeft +
lLabel1.spWidth + 3;
lLabel2.spTop := 2;
lLabel2.Caption := 'Company Name';
```

These two labels will print at the top of each page. The 'sp' in the spLeft and spTop properties refers to screen pixels. All positional properties of the components within a report are expressed in the units of the report itself (i.e. the value of the Report.Units property). Using the screen pixel version of these properties allows us to size and position components without concern for the current value of the Unit property.

7 Add data-aware components to the detail band.

```
lDBText1 := TppDBText.Create(Self);
lDBText1.Band := lReport.DetailBand;
lDBText1.spLeft := lLabel1.spLeft;
lDBText1.spTop := lLabel1.spTop;
lDBText1.DataPipeline := lDataPipeline;
lDBText1.DataField := 'CustNo';
lDBText2 := TppDBText.Create(Self);
lDBText2.Band := lReport.DetailBand;
lDBText2.spLeft := lLabel2.spLeft;
lDBText2.spTop := lLabel2.spTop;
lDBText2.DataPipeline := lDataPipeline;
lDBText2.DataField := 'Company';
```

These components are positioned in the detail band directly below their corresponding label components.

8 Add a page number and timestamp to the footer band.

```
lSysVar := TppSystemVariable.Create
↳ (Self);
lSysVar.Band := lReport.FooterBand;
lSysVar.VarType := vtPrintDateTime;
lSysVar.spLeft := 2;
lSysVar.spTop := 2;
lSysVar := TppSystemVariable.Create.
↳ (Self);
lSysVar.Band := lReport.FooterBand;
lSysVar.VarType := vtPageNoDesc;
lSysVar.Alignment := taRightJustify;
lSysVar.spLeft :=
↳ (lReport.PrinterSetup.PageDef.spPrintableWidth
↳  -lSysVar.spWidth) -2; lSysVar.spTop := 2;
```

Notice how the second system variable component is right-justified. The AutoSize property defaults to True for system variables, so we need only to set the Alignment property and component position. Here we set spLeft so that the component is flush with the right margin of the page (less 2 pixels for spacing). Because it is right-justified, this component will expand to the right when it prints. The PrinterSetup.PageDef object contains all of the dimensions of the page. The spPrintableWidth property contains the width of the paper less the left and right margins (in screen pixels).

9 Preview the report.

```
lReport.Print;
```

The DeviceType property of a report defaults to 'Screen,' so calling print here causes the Print Preview form to be displayed.

10 Free the report.

```
lReport.Free;
```

The ModalPreview property of the report defaults to True; therefore, this line of code will only fire after the Print Preview form has been closed. When the report is freed, it will automatically free the bands and components it contains.

11 Free the data access components.

```
lTable.Free;
lDataSource.Free;
lDataPipeline.Free;
```

The data access components are not freed by the report, and so must be freed separately.

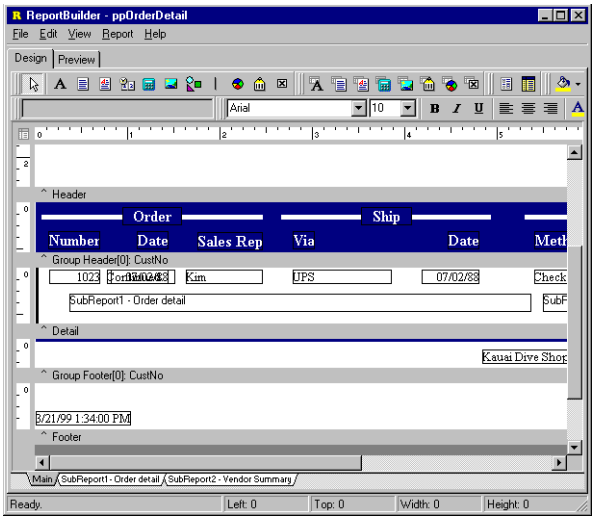
DESIGN

DESIGN

The Report Designer

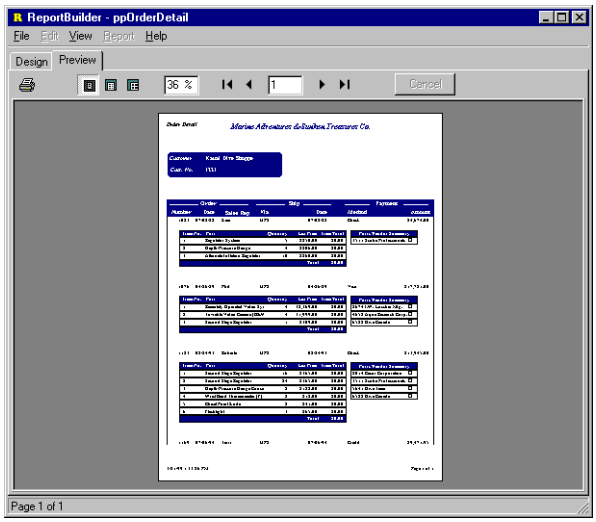
Design Tab

The design workspace is where the report layout is created. This workspace contains all of the menus, toolbars, and dialogs that make up the Report Designer.



Preview Tab

The Preview workspace shows a representation of the report as it will appear when printed to the printer. The iterative process of perfecting a report is generally accomplished by moving back and forth between the Preview and Design tabs of the Report Designer.



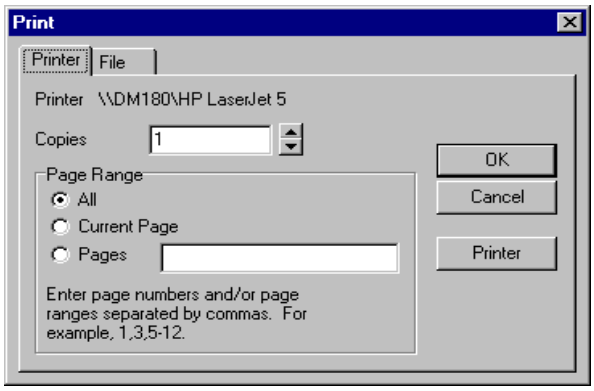
Dialogs

Print Dialog

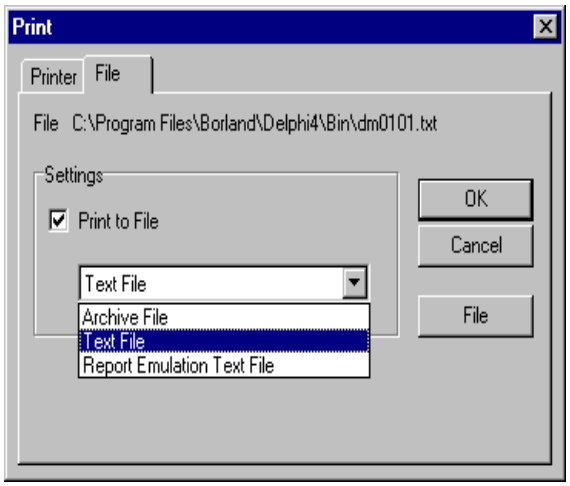
The Print Dialog is automatically displayed when the report is sent to the printer, allowing you to select the pages, number of copies, and printer for the report. When the AllowPrintToFile or AllowPrintToArchive properties of the Report are set to True, this dialog displays additional print to file options.

Print Dialog

1 The print job settings can be set via the standard dialog.



2 Dialog with the print to file options.

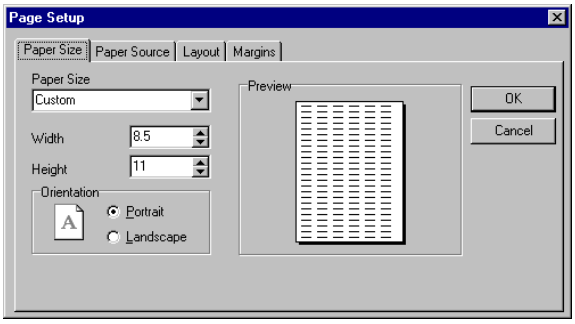


Page Setup Dialog

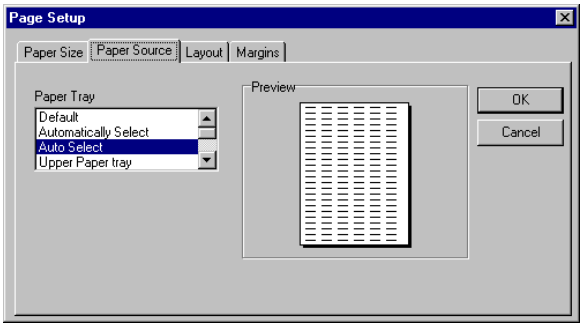
The Page Setup dialog can be accessed from the File | Page Setup menu option of the Report Designer. You can set the following properties from within the Page Setup dialog:

Print Dialog

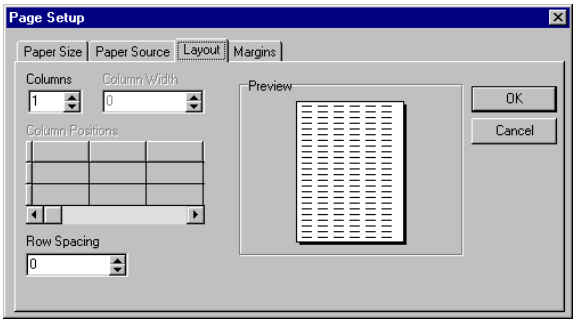
1 Paper size and orientation



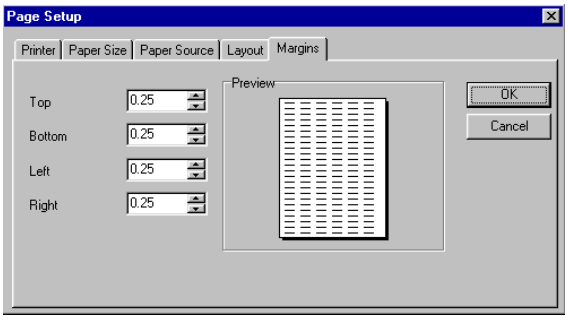
2 Paper Source



3 Layout for Columnar Reports

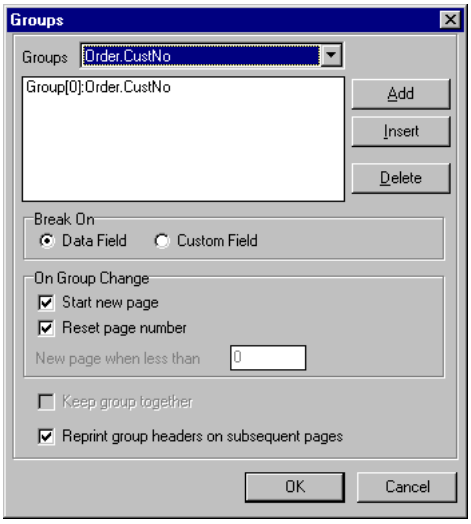


4 Margins



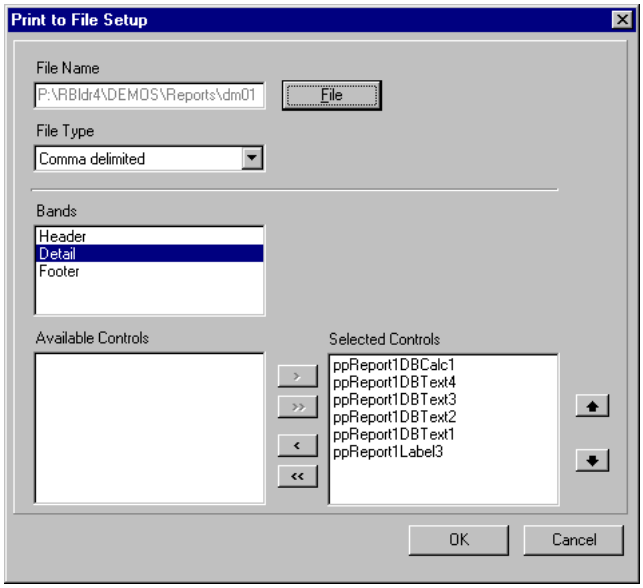
Groups Dialog

The Groups dialog is accessible via the Report | Groups menu option of the Report Designer. You can separate your report into different sections using groups. A number of options are available to control the behavior of each group. For example, you may want each group to start on a new page or to reprint the group header when the group continues on additional pages. Another powerful feature is the Keep group together option, which can be used to ensure that all of the information for a group fits on a page.



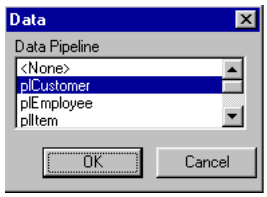
Print to File Setup Dialog

The Print to File Setup dialog is accessible via the File | Print to File Setup menu option of the Report Designer. This dialog is used to specify the format and content of the ASCII file that will be created if the report is printed to file.



Data Dialog

The Data dialog can be accessed from the Report | Data menu option of the Report Designer. It can be used to specify the data pipeline for the report.



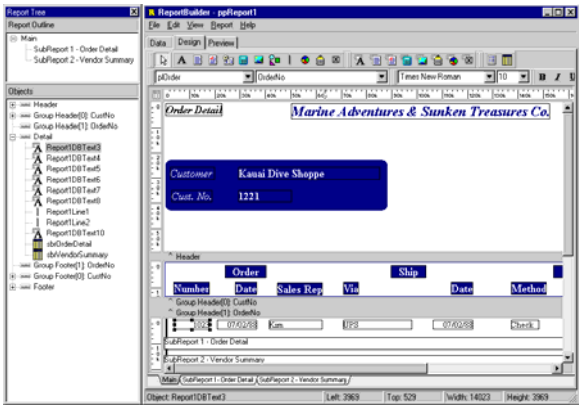
Toolbars

Overview

The various toolbars accessible from the design workspace are documented in this section. The toolbars are dockable and follow the Office97 interface style. The toolbars are accessible from the View | Toolbars menu option of the Report Designer or by right-clicking on the docking area at the top of the Report Designer.

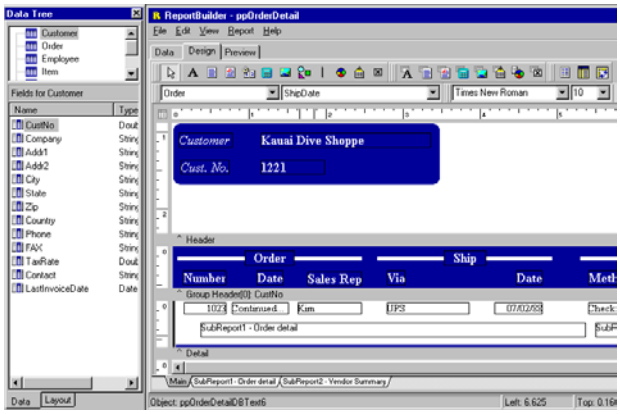
The Report Tree

To access this tool window, select the View | Toolbars | Report Tree menu option from the Report Designer main menu. This tool window is dockable on the left and right sides of the Report Designer. It can be used to organize components within each band. Components selected in the Report Tree are selected in the report layout. The upper portion of the Report Tree shows the main report object and any subreports nested within it. This feature can be helpful for organizing your subreports.



The Data Tree

To access this tool window, select the View | Toolbars | Data Tree menu option from the Report Designer main menu. This tool window is dockable on the left and right sides of the Report Designer. It can be used to create components within any band. Simply select a set of fields and drag the selection into the band. A set of corresponding data-aware components will be created.



Standard Component Palette

To access this toolbar, select the View | Toolbars | Standard Components menu option from the Report Designer main menu. This toolbar will assist in creating the most commonly used report components.

Label
Displays text. Assign the Caption property to control the text value. You can have the label resize automatically to fit a changing caption if you set the AutoSize property to True.

Memo
Prints multiple lines of plain text in a report. To set the value, assign a string list to the Lines property. To dynamically resize the memo during printing, set the Stretch property to True. Use the ShiftRelativeTo property to define dynamic relationships with other stretchable objects.

RichText
Prints formatted text. To set the value, assign the RichText property or use the LoadFromFile orLoadFromRTFStream methods. Use the ShiftRelativeTo property to define dynamic relationships with other stretchable objects. At design-time you can use the ReportBuilder's built-in RTF Editor to load, modify, and save rich text data stored in files.

SystemVariable
Displays common report information such as page number, page count, print date and time, date, and time. The type of information displayed is controlled by the VarType property. The format is controlled by the DisplayFormat property.

Variable
Used for calculations via an Object Pascal event handler assigned to the OnCalc event or a RAP event handler assigned to the OnCalc event. Access the Calculations dialog (via the speed menu) or the Calc tab of the Report Designer to code a RAP calculation for this component.

Image
Displays bitmaps and windows metafiles in reports. Assign the Picture property of this component in order to place an image in your report. Use the Report Designer's built-in picture dialog to load images at design-time.

Line
Displays single and double lines (either vertical or horizontal.) Set the Style property to control whether the line is single or double. Set the Weight property to control the line thickness in points. Set the Position property to control whether the line is vertical or horizontal.

Shape
Prints various shapes (squares, rectangles, circles, ellipses). Set the Shape property to select a type of shape. Use the Brush and Pen properties to control the color and border respectively.

TeeChart
Displays standard (non-data-aware) TeeCharts. This component enables you to use TeeCharts inside the Report Designer. You can access the TeeChart editor via a popup menu.

BarCode
Renders barcodes. The string value assigned to the Data property is encoded based on the BarCodeType. If the data to be encoded is in a database, use DBBarCode. The following symbologies are supported: Codabar, Code 128, Code 39, EAN-13, EAN-8, FIM A,B,C, Interleaved 2 of 5, PostNet, UPC-A, UPC-E.

CheckBox
Displays a checkbox using the WingDings font.

Data Component Palette

To access this toolbar, select the View | Toolbars | Data Components menu option from the Report Designer main menu. This toolbar will assist in creating data-aware report components.

DBText
Displays values from all types of database fields. Use the DisplayFormat property to format the value.

DBMemo
Prints plain text from a memo field of a database table. This control will automatically word-wrap the text. Set the Stretch property to True and the component will dynamically resize to print all of the text. Use the ShiftRelativeTo property to define dynamic relationships with other stretchable objects.

DBRichText
Prints formatted text from a memo or BLOB field of a database table. This control will automatically word-wrap the text. Set the Stretch property to True and the component will dynamically resize to print all of the text. Use the ShiftRelativeTo property to define dynamic relationships with other stretchable objects.

DBCalc
Used for simple database calculations (Sum, Min, Max, Count, and Average). The value can be reset when a group breaks using the ResetGroup property.

DBImage
Prints bitmaps or windows metafiles, which are stored in a database BLOB field.

DBBarCode
Renders barcodes based on the BarCodeType and the value supplied via the DataField property. The following symbologies are supported: Codabar, Code 128, Code 39, EAN-13, EAN-8, FIM A,B,C, Interleaved 2 of 5, PostNet, UPC-A, UPC-E.

DBTeeChart
Allows data-aware TeeCharts to be placed within a report.

DBCheckBox
Displays a checkbox based on the value of the field specified in the DataField property. This component can be used with a Boolean field (or any other type of field via the BooleanTrue, BooleanFalse properties).

Advanced Component Palette

To access this toolbar, select the View | Toolbars | Advanced Components menu option from the Report Designer main menu. This toolbar will assist in creating advanced report components.

Region

Logically groups components together. Use the ShiftRelativeTo property to move the region in relation to another dynamically resizing component (such as Memo, RichText or child-type SubReport).

SubReport

Handles multiple master details, creates side-by-side reporting effects, and hooks reports together as one. If you need a report to print within the context of a band, use a child-type subreport. If you need to hook reports together, use a section type subreport. The PrintBehavior property determines the subreport type.

CrossTab

Allows you to generate a set of calculations that summarizes the data from a database table. It displays the calculations in a grid format.

Standard Toolbar

To access this toolbar, select the View | Toolbars | Standard menu option from the Report Designer main menu. This toolbar will assist with saving the report layout, accessing the print and print preview options, and accessing the cut and paste operations.

New Report

Creates a blank report layout.

Open Report

Displays the Open dialog, allowing you to open an existing report layout.

Save Report

Saves a report layout to file.

Page Setup

Displays the Page Setup dialog, allowing you to set the paper size and configure the layout for the report.

Print

Displays the Print dialog before sending the report to the printer.

Print Preview

Displays the Print Preview window.

Cut

Cuts the currently selected components into the clipboard.

Copy

Copies the currently selected components into the clipboard.

Paste

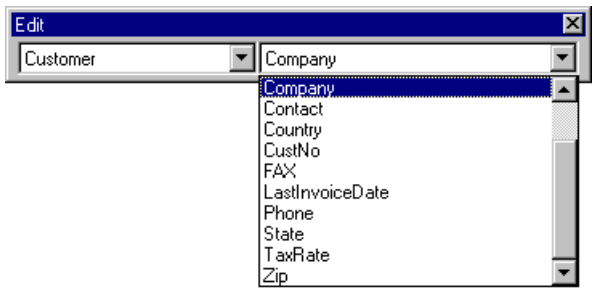
Pastes the components in the clipboard into the report.

Edit Toolbar

To access this toolbar, select the View | Toolbars | Edit menu option from the Report Designer main menu. This toolbar will assist in setting the most important property or properties for the currently selected component.

1 No component selected

2 Data-aware component selected

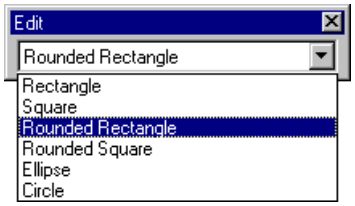


This configuration allows the data pipeline and data field for the component to be set. The drop-down list on the left shows the data pipeline. The drop-down list on the right shows the field name.

3 Label component selected

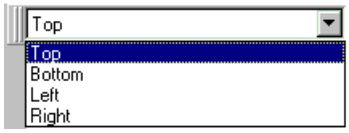
Here a label component has been selected in the Report Designer. The Edit toolbar displays an edit box from which the label's caption can be set.

4 Shape component selected



Here a shape component has been selected in the Report Designer. The Edit toolbar displays the different shape types.

5 Line component selected



This configuration allows you to move the line to the top, bottom, left, or right within the line's selection handles.

Format Toolbar

To access this toolbar, select the View | Toolbars | Format menu option from the Report Designer main menu. This toolbar will assist with setting the font and colors. It will also assist with layering the components.

Font Name

Selects the font name for textual components. Use TrueType fonts (indicated by a icon) when possible. These render well on both the screen and printer. If you are using a dot-matrix printer, the print driver may supply printer fonts (indicated by a icon) that you can use to speed up the printing of the report. Finally, fonts that have no icon to the left of the font name are screen fonts and should not be used in reports where WYSIWYG is required.

Font Size

Selects the font size. You can also type in this box to set the font size exactly.

Bold

Sets the font to bold.

Italic

Sets font to italic.

Underline

Sets font to underline.

Left Justify

Left justifies the text in the component.

Center

Centers the text in the component.

Right Justify

Right justifies the text in the component.

Font Color

Sets the font color.

Highlight Color

Sets the background color of the textual component.

Bring to Front

Brings the component to the front. The components in the front print last, and the components in the back print first. Use the Report Tree to see the exact layering of components within the band.

Send to Back

Sends the component to the back. The components in the front print last, and the components in the back print first. Use the Report Tree to see the exact layering of components within the band.

Align or Space Toolbar

To access this toolbar, select the View | Toolbars | Align or Space menu option from the Report Designer main menu. This toolbar will assist in positioning components relative to one another and relative to the band in which they appear.

Align Left Edges
Aligns a group of components with the leftmost position of the component that was selected first.

Align Horizontal Centers
Centers a group of components based on the horizontal center of the component that was first selected.

Align Right Edges
Aligns a group of components with the rightmost position of the component that was selected first.

Align Top Edges
Aligns a group of components with the topmost position of the component that was selected first.

Align Vertical Centers
Aligns a group of components based on the vertical center of the component that was first selected.

Align Bottom Edges
Aligns a group of components with the bottommost position of the component that was selected first.

Space Horizontally
Spaces a set of components based on the leftmost position of the first component selected and the rightmost position of the last component selected.

Space Vertically
Spaces a set of components based on the topmost position of the first component selected and the bottommost position of the last component selected.

Center Horizontally in Band
Centers a component horizontally within a band.

Center Vertically in Band
Centers a component vertically within a band.

Size Toolbar

To access this toolbar, select the View | Toolbars | Size menu option from the Report Designer main menu.

Shrink Width
Determines the minimum width of all the selected components, and then sets the width of the components to that value.

Grow Width
Determines the maximum width of all the selected components, and then sets the width of the components to that value.

Shrink Height
Determines the minimum height of all the selected components, and then sets the height of the components to that value.

Grow Height
Determines the maximum height of all the selected components, and then sets the height of the components to that value.

Nudge Toolbar

To access this toolbar, select the View | Toolbars | Size menu option from the Report Designer main menu.

Nudge Up
Moves all selected components one pixel up.

Nudge Down
Moves all selected components one pixel down.

Nudge Left
Moves all selected components one pixel to the left.

Nudge Right
Moves all selected components one pixel to the right.

Draw Toolbar

To access this toolbar, select the View | Toolbars | Draw menu option from the Report Designer main menu. This toolbar will assist in setting the colors and borders of components.

Fill Color

For shapes, lines, and region components only. Sets the Brush.Color property. To set the color of a textual component, check the Highlight Color action of the Format toolbar.

Line Color

For shapes, lines, and region components only. Sets the Pen.Color property.

Line Thickness

For use with a Line component only. Sets the Weight property.

Line Style

For use with a Line component only. Sets the Pen.Style property.

Drag and Drop Support

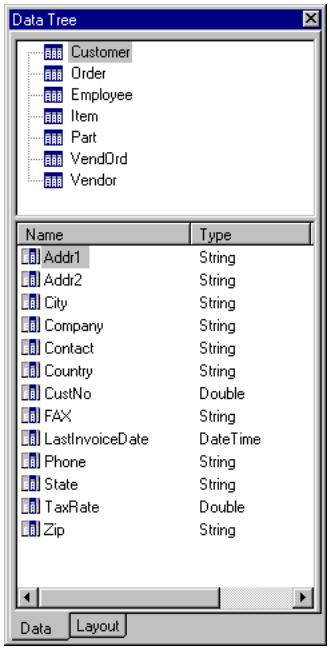
Overview

ReportBuilder contains a Report Wizard that allows you to quickly create an entire report layout. This is great for generating an entire report, but what if you need to create only a portion of a complex report? Drag and drop functionality is an ideal solution for this problem because it allows you to create a set of components within the context of an existing report layout. In ReportBuilder, drag and drop support is provided via the Data Tree.

The Data Tree has two tabs. In the top tree view, the Data tab contains a list of data pipelines to which the report has access. In the bottom list view, all of the fields for the currently selected data pipeline are displayed. Fields can be selected from the bottom list view and dragged to any part of the report layout. The data-aware component that is appropriate for the given field will then be created along with a label and border.

The second tab of the Data Tree is entitled Layout. This tab allows you to control drag-and-drop behavior. A label and border are created for each data-aware component by default. You can turn the label and the border off, control the color of the label or border, and control the font of the label and data-aware component from this tab. Once you've set the drag-and-drop behavior, it will be retained for future design sessions.

Data Tree Tabs



Data Tree - Data tab

Data Tree - Layout tab

The Report Wizard

Overview

The Report Wizard is one of the many parts of ReportBuilder that reflects a level of professionalism and attention to detail found in no other reportou will be able to quickly recognize and use the ReportBuilder Report Wizard.

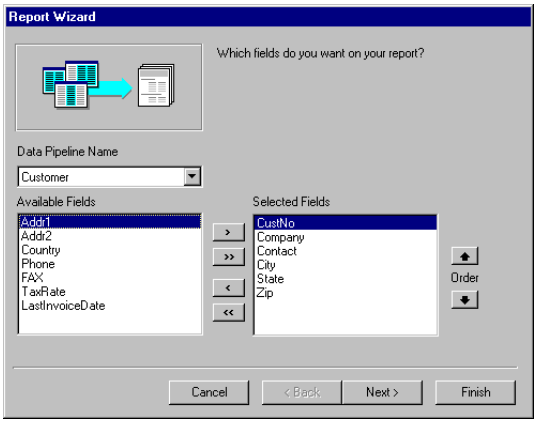
The Report Wizard can be accessed via the File | New menu option of the Report Designer. A series of screens are presented, each requesting information about the report. When the last page is reached, either a preview or design option can be selected. When the 'Finish' button is clicked, it causes a report to be created and displayed as requested.

Report Wizard: Create a Simple Report-

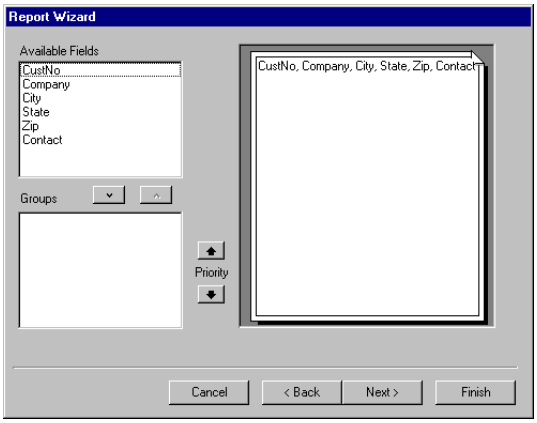
The following screen shots step through the creation of a simple report via the Report Wizard.

Create a Simple Report

1 Select the fields.



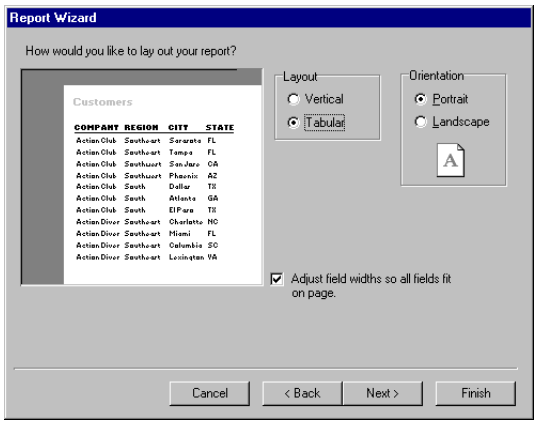
2 Skip the groups page.



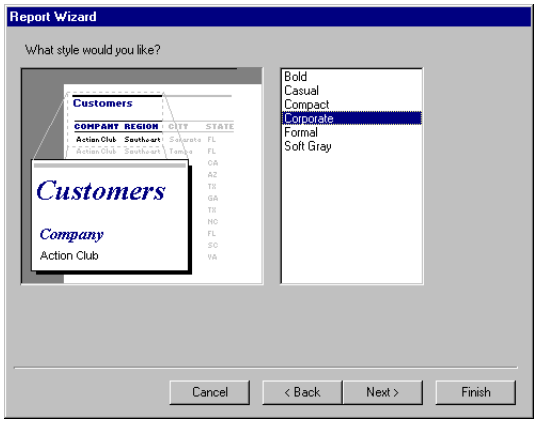
Report Wizard: Create a Simple Report - cont.

Create a Simple Report - cont.

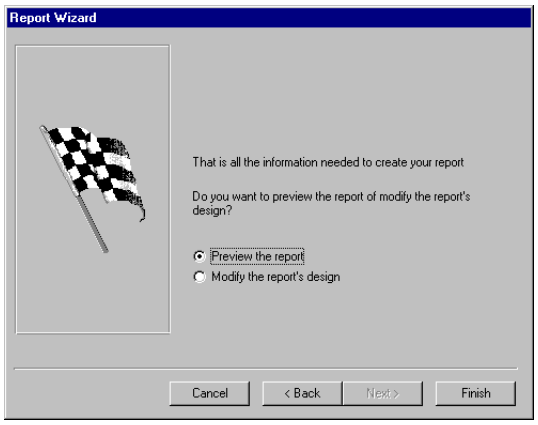
3 Select the layout.



4 Select the style.



5 Select Design or Preview and Finish.

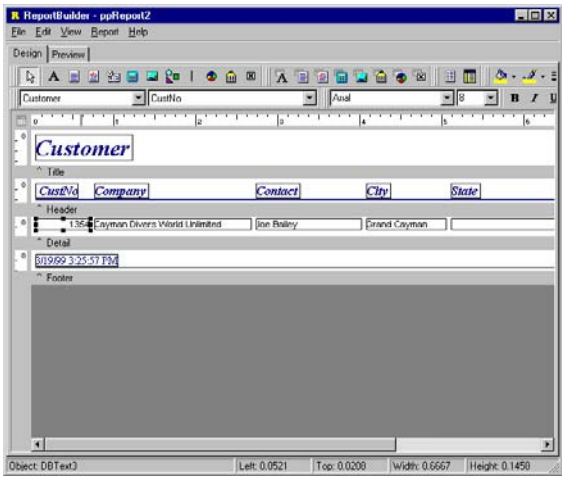


Report Wizard: Create a Simple Report - cont.

Create a Simple Report - cont.

Report as it appears in the Preview workspace

Report as it appears in the Design workspace

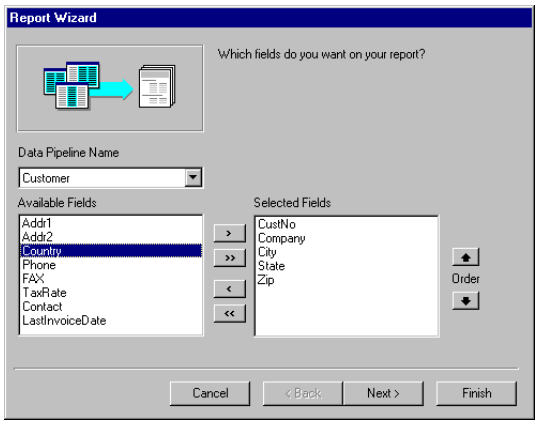


Report Wizard: Create a Group-Based Report

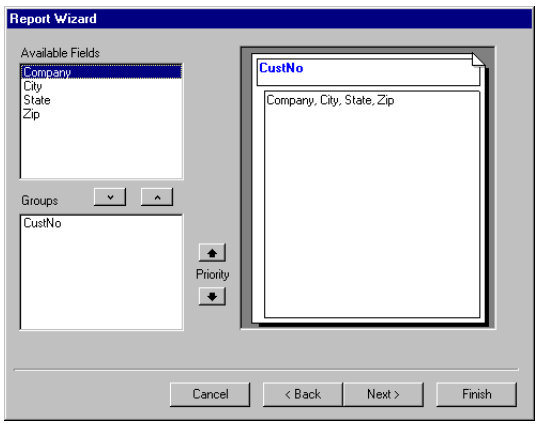
The following screen shots steps through the creation of a group-based report via the Report Wizard.

Create a Group-Based Report

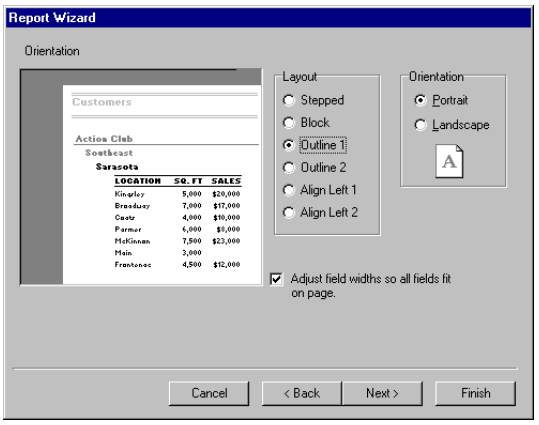
1 Select the fields.



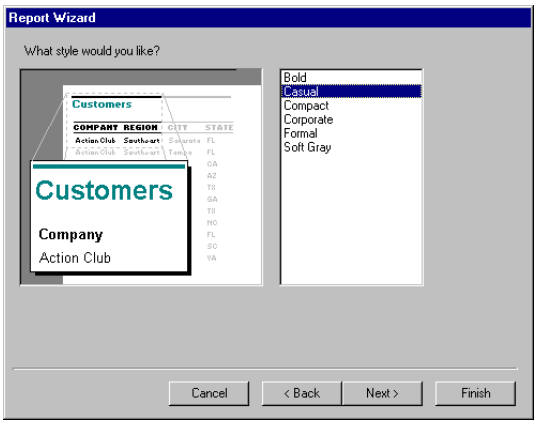
2 Select a group field.



3 Select the layout.



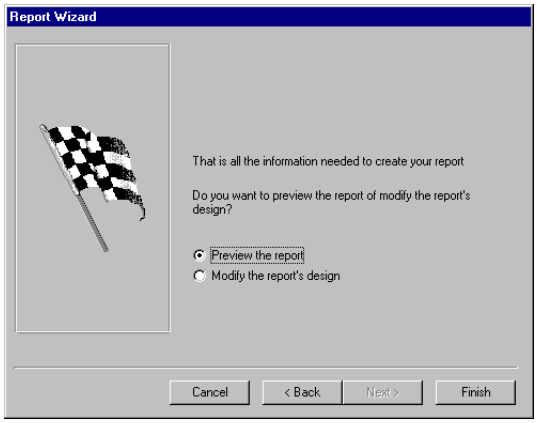
4 Select the style.



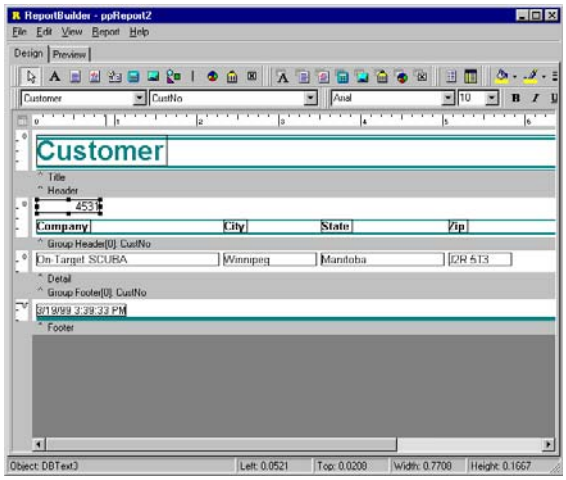
Report Wizard: Create a Group-Based Report - cont.

Create a Group-Based Report cont.

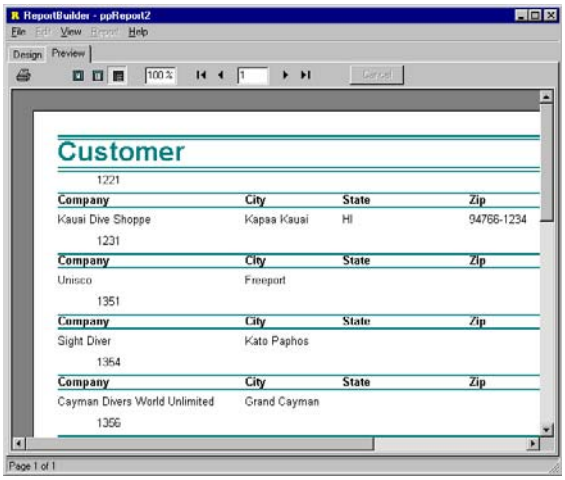
5 Select Design or Preview and Finish.



Report as it appears in the design workspace.



Report as it appears in the Preview workspace.



PRINT

PRINT

Introduction

Generation

Report output is primarily generated via a call to the Print method of the report object.

1 Print to Screen

The following code would cause the Print Preview form to be displayed and the first page of the report to be presented in this form:

```
uses
   ppTypes;

ppReport1.DeviceType := dtScreen;
ppReport1.Print;
```

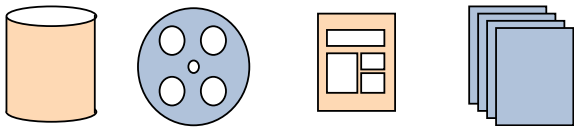
2 Print to Printer

This code would cause the Print dialog to be displayed. The report would then be sent to the printer:

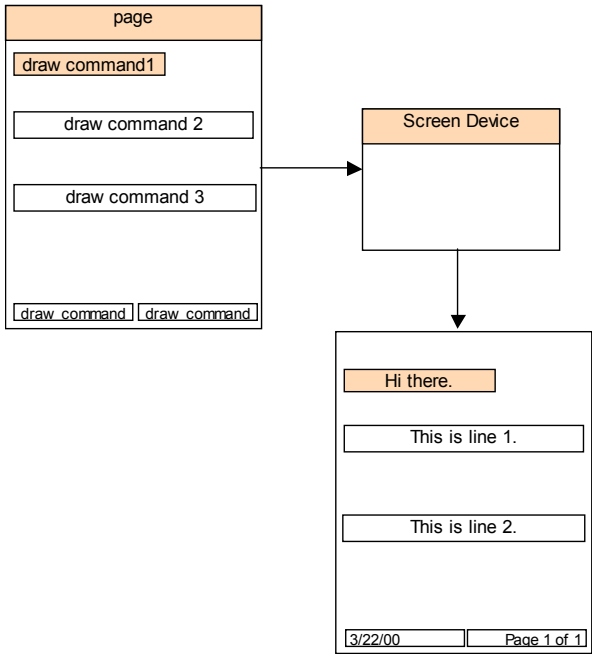
```
uses
   ppTypes;

ppReport1.DeviceType := dtPrinter;
ppReport1.Print;
```

As you can see, it is quite easy to generate a report using ReportBuilder. In order to use some of the more advanced features related to report output, you will need to understand a little bit more about what is happening behind the scenes.

As you may already know, report output is one of the elements of the reporting equation. The reporting equation was described in the introduction to ReportBuilder and is as follows:



The native report output in ReportBuilder takes the form of page objects. Page objects contain draw commands, which describe what the page contains. Page objects are sent to devices, which then convert them to a format appropriate for the device. The following diagram shows page objects that are sent to the ScreenDevice and then converted to a bitmap for display in the Print Preview form.

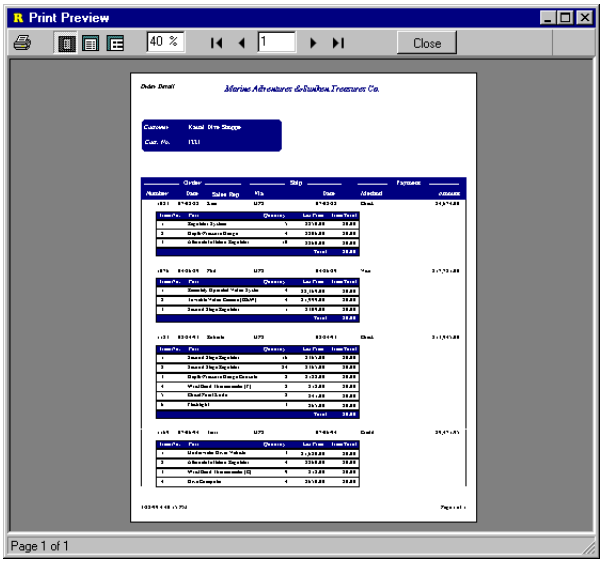


Several device classes are delivered standard with ReportBuilder. Each device class supports a different physical device or file format, but the purpose of the device remains the same: to convert the native report output of ReportBuilder to the format appropriate for the given device. The remainder of this section describes the different devices and their capabilities.

Previewing

Print Preview

The most high-profile and oft-used form of report output is contained in the Print Preview form. The Print Preview form is launched automatically when the Print method of the report component is called (and the DeviceType property has been set to 'Screen'). The Print Preview form is pictured below.



From this form, the user can preview any page of the report, zoom the page to different levels, and print the report to the printer.

While the Print Preview form provided with ReportBuilder looks and performs like a truly professional user-interface should, you may want to customize it to emulate the look and feel of the rest of your application. This customization can be done by creating a special form that descends from the CustomPreviewer class and registering it as the 'official' print preview form. This process is fully described in the 'Building a Reporting Application' tutorial.

Custom Printing Settings

Overview
ReportBuilder contains a variety of properties that allow you to control various aspects of the print job. The following properties are contained in the PrinterSetup property of the Report object and can be configured via the Object Inspector or the File | Page Setup dialog of the Report Designer.

BinName
The name of the bin (paper tray) containing the paper on which the report will be printed. A common use of this property is to set it to Manual Feed on mailing label reports. This way, you can provide a means to load the special label paper without manually setting the printer.

Collation
When printing multiple copies of a report, this property determines whether each copy prints as a separate set of pages.

Copies
The number of copies of the report that should be printed.

DocumentName
The name of the print job to be used when the report is printed - this is the name as it will appear in the Print Manager.

Duplex
For printers that support duplex, this setting determines the type of duplex printing that will occur.

Margins
The margin properties determine the left, top, right, and bottom margins of the report.

Orientation
Either landscape or portrait.

PaperHeight & PaperWidth
The dimensions of the selected paper, based on the value of the PaperName property. If these values are entered manually and no corresponding paper size is found, the PaperName is set to 'Custom'.

PaperName
The name of the paper type, such as 'Letter', or 'A4'.

PrinterName
The name of the printer to which the report should be printed. This property defaults to 'Default', which will cause ReportBuilder to use the default printer declared on your system.

Because all ReportBuilder reports contain a PrinterSetup object, it is possible to use section-type subreports to achieve some fairly incredible levels of customization of the print behavior. For example, different pages of a report can print from different bins of the same printer or print to different printers entirely. Some of these features are showcased in the printer setting examples provided with ReportBuilder (examples 121-124).

Report Archiving

Overview

Archiving is a process by which an instance of a report output is saved for future use. This capability is most often used to save a snapshot of a report as it appeared on a given day and at a certain time. If the data that was used to generate the report changes, then a representation of the report as it appeared at the given moment is still available. Archiving can also be used to generate daily or monthly reports during off-peak hours and then enable users to preview or print the reports at a later time.

1 Archiving a Report

A Print directly to archive.

You can print a report directly to archive with the following code:

```
uses
   ppTypes;

   ppReport1.ShowPrintDialog := False;
   ppReport1.DeviceType := dtArchive;
   ppReport1.ArchiveFileName := 'c:\test.raf';
   ppReport1.Print;
```

In this code, the Print dialog is hidden; the device type is set to 'Archive'; the archive file name is specified, and the report output is generated. Archive files are usually given the file extension 'raf', which stands for Report Archive File.

B Allow end users to print to archive.

In order to give the end user the opportunity to print the report to archive whenever the Print method is called, use the following code:

```
uses
   ppTypes;

ppReport1.AllowPrintToArchive := True;
ppReport1.ShowPrintDialog := True;
ppReport1.DeviceType := dtPrinter;
ppReport1.ArchiveFileName := 'c:\test.raf';
ppReport1.Print;
```

When the print method is called, the Print dialog will be displayed. Because AllowPrintToArchive has been set to True, the Print dialog will contain two notebook tabs: 'Printer' and 'File'. If the end user clicks the 'File' tab, the archive file name will be defaulted.

2 Using the ArchiveReader

Once archive files have been created, they can be read by the ArchiveReader component. The ArchiveReader component has a set of capabilities similar to the Report component. In fact, these two components descend from the same common ancestor. You can print the contents of an archive file to a standard Print Preview window with the following code:

```
uses
   ppTypes;

ppArchiveReader1.ArchiveFileName :=
↳ 'c:\test.raf';
ppArchiveReader1.DeviceType := dtScreen;
ppArchiveReader1.Print;
```

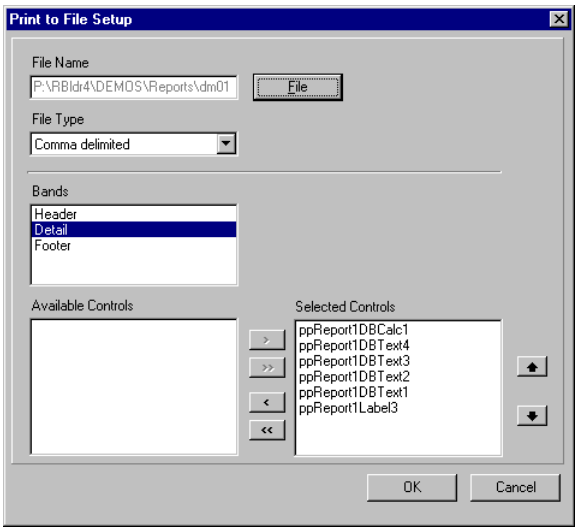
Print to ASCII Text

Overview

The TextFile device in ReportBuilder converts report output into an ASCII text file. The intent of this device is to create structured data that can be imported into another application, such as a spreadsheet. The following text file formats are supported:

- comma-delimited
- tab-delimited
- fixed length records
- custom-delimited

Because report output is essentially free-form and can contain many non-textual components, ReportBuilder provides a way for you to tightly control the format of the text file generated via this device. This control is provided by the Print to File Setup dialog (accessible from the File | Print to File Setup menu option of the Report Designer). This dialog is pictured below:



This dialog allows you to set the default file name, select the file format, and specify the textual components that will be saved to the file and the order that they will be saved. When utilizing the TextFile device, it is mandatory that these settings are completed. If no components have been selected, then a blank text file will result.

Print directly to the ASCII text file

Once a report has been configured for ASCII text file output, the following code will generate the report directly to the file:

```
uses
   ppTypes;

ppReport1.ShowPrintDialog := False;
ppReport1.DeviceType := dtTextFile;
ppReport1.Print;
```

Allow the end user to print to the ASCII text file

The following code will give the end user the option of printing to text when the print method is called:

```
uses
   ppTypes;

ppReport1.AllowPrintToFile := True;
ppReport1.DeviceType := dtPrinter;
ppReport1.ShowPrintDialog := True;
ppReport1.Print;
```

The report component has an OnSaveText event that fires each time a textual component is saved to the text file. This event can be used to customize the output generated to the text file. An example of what can be achieved via this event is provided in example 106 (..\RBuilder\Demos\Reports\dm0106.pas).

Report Emulation Text File

Overview

The TextFile device discussed in the previous topic is used for ASCII text file output. This type of device is excellent for creating portable data from report output. However, the formatting of the report is lost in a tightly defined comma or tab-delimited data format. When text file output that retains the spacing of the original report is needed, the ReportTextFile device can be used. Text files generated with this device are useful for sending as e-mail attachments or directly to dot-matrix printers for high-speed output.

The ReportTextFile device does not require any special configuration in advance. All textual components (except memos and richtext) will be printed in the text file.

Print directly to report emulation text file

The following code sends a report to a report emulation text file:

```
uses
   ppTypes;

ppReport1.ShowPrintDialog := False;
ppReport1.DeviceType := dtReportTextFile;
ppReport1.TextFileName := 'c:\test.txt';
ppReport1.Print;
```

Allow the end user to print to a report emulation text file

The following code would give the end user the option of printing to a report emulation text file when the print method is called:

```
uses
   ppTypes;

ppReport1.AllowPrintToFile := True;
ppReport1.TextFileName := 'c:\test.txt';
ppReport1.DeviceType := dtPrinter;
ppReport1.ShowPrintDialog := True;
ppReport1.Print;
```

In order to get the free-form text of the report output to map correctly to the character grid of a text file, it is sometimes necessary to tweak the positions of the components within the report layout. An example of a report that has been configured for use with the Report Emulation Text File device is provided in example 107 (..\RBuilder\Demos\Reports\dm0107.pas).

RTF, HTML, and Other Formats

Overview

By utilizing the highly extensible device architecture of ReportBuilder, you can create your own device class descendants for converting the native output of ReportBuilder into any file format. Some developers have already done just that, and are offering device classes for some of the most popular file formats. These additional devices integrate seamlessly into ReportBuilder.

TExtraDevices

by James Waler

Provides file device support for RTF, HTML, Excel, Lotus, and Quattro Pro. A Demo is available for download. Contact James Waler directly at www.waler.com for more details.

PsRBExportDevices

by Pragnaan Software

Provides file device support for PDF, HTML, RTF, Excel, JPEG, GIF, BMP, EMF and WMF. A demo is available for download. Contact Pragnaan Software at www.pragnaan.com for more details.

DEPLOY

DEPLOY

Introduction

Overview

There are two totally different but vital issues involved in deciding how you will deploy your application:

1 How to deploy ReportBuilder as part of your application

2 How to deploy the reports created with ReportBuilder as part of your application

Application Deployment

When creating your application exe, you can choose to either compile everything into one executable, or you can choose to distribute the packages provided with Delphi, ReportBuilder, or any other components you are utilizing in your application as separate files. The latter option is referred to as 'distributing with packages' (packages here refers to a Delphi compatible DLL) and results in a smaller, more efficient executable file.

These two approaches for deploying your application are discussed later in this section.

Report Deployment

The second part of deployment involves determining how to deploy the reports created with ReportBuilder as part of your application. ReportBuilder provides the most flexible deployment options of any reporting product on the market. The four most commonly used deployment strategies are as follows:

1 Compile the reports into the executable.

Allow the report component to reside in an invisible form along with the necessary data access objects. When it is time to print the report, instantiate the form and call the Print method. A slight variation of this architecture would be to use a data module as a container for the data access objects. This method requires the form to contain the ppReport component.

2 Compile the reports into a package.

This strategy is similar to option 1, but the report forms are compiled into a package rather than an executable. The advantage of this approach is that it optimizes the size of the executable and dynamically loads the package containing the reports as needed.

3 Distribute the report template files.

Place all the data access components for the reports in a data module. Place a single report component on a form that 'uses' the data module. Create all of the report layouts via this one report component, saving each one down to a separate report layout file. When it is time to print the report, load the report from file and call the Print method.

4 Distribute a report database.

Use all of the same steps as option 3, but save the report definitions to a database BLOB field instead of separate files. With this approach, you can save all of your report definitions in one database table.

A full-featured implementation of Option 1 is provided in the 'Building a Reporting Application' tutorial. Option 1 has the advantage of requiring the least code. The main advantage of options 3 and 4 is that you can change report layouts and redeploy them to your end users without compiling your executable. Because Option 1 compiles the report layouts into the executable, it requires a recompile any time a report must be changed. In terms of report layout load time or print speed, all options are essentially equal.

Report Templates

Overview

'Report Template' refers to the binary image created when a report layout is saved to a file or to a database BLOB field. You can use report templates to free your report definitions from the confines of the executable in which they are being utilized. Therefore, if you need to make modifications to a report, you can do so without recompiling and redeploying your entire application. Simply send the new report layout file or database table containing the report layouts to your users.

If you save report layouts to files or database BLOB fields, then you can use them to create a versioning system for reports. This way, the original reports supplied with your application are never modified, but end users can create versions of these reports. In order to allow end users to modify the reports via the Report Designer, ReportBuilder Enterprise or Pro is required.

Report templates leverage technology already present in Delphi. The same logic used to save the state of objects as configured on a Delphi form (and stored in a dfm file) is used to save the components of a report layout (in an rtm file). In order to view the content of an rtm file, set the Report.Template.Format to ftASCII and save the report. Then open the resulting rtm file in the Delphi code editor. You will be able to see the structure of this file format.

File-based Templates

You can load and save report layouts to files using the Report Designer. You can also load and save layouts programmatically.

1 Using the Report Designer

Set the Report.Template.SaveTo property to stFile (this is the default value).

A Saving a Report Layout

Select the File | Save As... menu option from within the Report Designer. The standard Windows file save dialog will be displayed, enabling you to save the report template to a .rtm file.

B Loading a Report Template

Select the File | Open... menu option from within the Report Designer. The standard Window file open dialog will be displayed, enabling you to select and open a .rtm file.

2 Programmatically

A Saving a Report Layout

The following code saves a report in the 'test.rtm' file:

```
ppReport1.Template.FileName := 'c:\test.rtm';
ppReport1.Template.SaveToFile;
```

B Loading a Report Template

The following code opens a report layout and prints the report:

```
ppReport1.Template.FileName := 'c:\test.rtm';
ppReport1.Template.LoadFromFile;
ppReport.Print;
```

Database Templates

You can save and load data-based report templates by using the Report Designer or programmatically.

Defining the Database Table

Report definitions can be stored to any database table with the following structure:

Name Char(40)
The name of the report is stored in this field.

Template BLOB or Memo
The report definition is stored in this field. If the format is binary, the field should be a BLOB; if the format is ASCII, the field should be a memo.

Connecting the Report object to the database

Once the database table has been defined, the next step is to connect the report to the database table. This task is accomplished by configuring the following properties of the Report.Template object. You can do this via the Delphi Object Inspector or programmatically:

```
Report.Template.DatabaseSettings.NameField
↳ := 'Name';
Report.Template.DatabaseSettings.TemplateField
↳ := 'Template';
Report.Template.DatabaseSettings.DataPipeline
↳ := plReports;
```

Loading and Saving Reports

1 Using the Report Designer

Set the Report.Template.SaveTo property to stDatabase.

A Saving a Report Layout

Select the File | Save As... menu option from within the Report Designer. A special Save dialog will be displayed, showing all reports in the given database table. You can name the report and then save it to the table.

B Loading a Report Template

Select the File | Save As... menu option from within the Report Designer. Once a report is saved, you can load it by accessing the File | Open dialog.

2 Programmatically

A Saving a Report Layout

The following code saves a report in the 'test.rtm' file:

```
ppReport1.Template.FileName := 'Order Summary';
ppReport1.Template.SaveToDatabase;
```

B Loading a Report Template

The following code opens a report layout and prints the report:

```
ppReport1.Template. DatabaseSettings.Name :=
↳ 'Order Summary';
ppReport1.Template.LoadFromDatabase;
ppReport.Print;
```

As an EXE

Overview

ReportBuilder can be deployed as part of your executable. Simply include the correct path to the compiled units (dcu files) of ReportBuilder in your Delphi Library Path. For example:

'c:\program files\borland\delphi5\rbuilder\lib'

The 'lib' directory contains the compiled units of ReportBuilder. When you compile your application, necessary dcu files from this directory will be linked into the resulting executable file. Executable files created this way are stand-alone in that they can be distributed successfully without any supporting files.

Applications containing ReportBuilder will not compile correctly if the library path is not configured properly. If there are directories in the Library path pointing at old versions of ReportBuilder or at the Source directory of ReportBuilder, problems will result.

Accessing the Delphi Library Path

1 Delphi 4 and 5

In order to access your Library Path in Delphi, select Tools | Environment Options from the Delphi main menu. The Environment Options dialog will be displayed. Click the Library tab. The Library Path is the first item. Click the dialog button to the left of the library path edit box. The directories that make up your library path will be displayed.

2 Delphi 3

In order to access your Library Path in Delphi 3, select Tools | Environment Options from the Delphi main menu. The Environment Options dialog will be displayed. Click the Library tab. The Library Path is displayed in a drop-down combobox near the bottom of this dialog. It is a good idea to scroll through this edit box character- by-character using the arrow keys when checking your library path, as highlighting the text in this edit box causes the cursor to jump to the end. This can cause you to miss duplicate or incorrect directories in the path.

As Packages

Overview

You can optimize the size of your executable by compiling your project with packages. To compile with packages, access the Project Options dialog, select the 'Packages' tab, and click on the 'Build with Packages' check box. The list of package names that appears to the left of this check box should include the following ReportBuilder run-time packages:

ReportBuilder run-time packages

rbRCL55	report components
rbBDE55	BDE data access components
rbUSER55	sample RCL components (checkbox)
rbTC51	standard TeeChart wrapper
rbTDBC51	data-aware TeeChart wrapper

International Language Support

Overview

ReportBuilder includes support for 12 built-in languages. The strings for each language reside in separate resource files. These files are located in the ..RBuilder\Languages directory, under a subdirectory named for the given language. Three letter codes are used to differentiate between the lanuage DLLs, which are installed in the Windows/System directory.

Languages

Language	Enumerated Type	Folder/File Extension
Default	lgDefault	dft
English	lgEnglish	eng
Custom	lgCustom	cst
Danish	lgDanish	dan
Dutch	lgDutch	nld
French	lgFrench	fra
German	lgGerman	deu
Italian	lgItalian	ita
Portuguese (Brazil)	lgPortugueseBrazil	ptb
Portuguese	lgPortuguese	ptg
Spanish	lgSpanish	esp
Spanish (Mexico)	lgSpanishMexico	esm
Swedish	lgSwedish	sve
Norwegian	lgNorwegian	nor

Only the languages that are explicitly selected during install will appear in the Languages folder. The 'cst' folder will always appear in the Languages folder because it represents the custom language type. The cst folder contains the resource files that correlate to the lgCustom enumerated type. The custom language can be modified to contain your own translation, which will then appear in the ReportBuilder user-interface when the Report.Language property is set to lgCustom.

The Default Language

When ReportBuilder is installed, the installation program allows you to specify a default language and any additional languages you wish to use. The installation will then install the appropriate files onto your system. The Report.Language property defaults to the lgDefault language type. This type correlates to the language resource files located in the RBuilder\Lib directory and the language DLLs located in the Windows\System directory (the DLLs with the .dft extension). The .res files are used when compiling stand-alone executables in Delphi. The DLLs are used to provide language support at Delphi design-time. When ReportBuilder installs, it copies these files to the appropriate directories.

The language DLLs that appear in Windows\System are listed below:

File Name	Description
rbPrint.dft	Strings used by ReportBuilder
rbIDE.dft	Strings used by the Report Designer

The resource files that appear in RBuilder\Lib are listed below:

File Name	Description
rbPrint.res	Strings used by ReportBuilder
rbIDE.res	Strings used by the Report Designer

You can change the default language used by ReportBuilder by copying the appropriate language files from the RBuilder\Languages directory to the RBuilder\Lib and Windows\System directories respectively. For the DLL files, make sure that you change the extension to .dft. The following copy commands will change the default language to Swedish when issued at the MS-DOS prompt (from within the RBuilder\Languages\sve directory):

```
copy rbPrint.res
c:\progra~1\borland\delphi5\rbuilder\lib
↳ copy rbIDE.res
c:\progra~1\borland\delphi5\rbuilder\lib
↳ copy rbPrint.sve
c:\windows\system\rbPrint.dft
↳ copy rbIDE.sve
c:\windows\system\rbIDE.dft
```

Note: If you are using Windows NT, you will need to copy the 'sve' files to Windows\System32.

Custom Translations

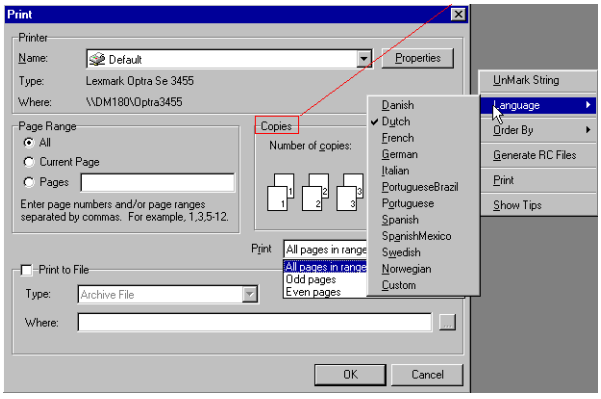
You can modify the .rc source files in the custom language folder in order to create your own custom translation. The rc files are named the same as their resource file counterparts:

File Name	Description
rbPrint.rc	Strings used by ReportBuilder
rbIDE.rc	Strings used by the Report Designer
rbDADE	Strings used by the Data Access Development Environment (the 'Data' tab)

You can create a custom translation by completing the following steps:

1 Run the Language Translation application.

Create new rc files using the ReportBuilder Language Translation application. This application displays the strings used within ReportBuilder along with an associated screen-shot. It provides an edit box in which the translation for each string can be entered. When you complete the translation, click the 'Generate RC Files' button, and generate each of the three rc files. The language translation app is pictured below.



2 Create new resource files.

Compile the rc files into resource files by issuing the following commands at the MS-DOS prompt (from within the 'cst' folder):

c:\progra~1\borland\delphi5\bin\brc32 -r rbPrint.rc
c:\progra~1\borland\delphi5\bin\brc32 -r rbIDE.rc
c:\progra~1\borland\delphi5\bin\brc32 -r rbDADE.rc

After these commands have executed, you should have three resource files in the cst folder: rbPrint.res, rbIDE.res, and rbDADE.res.

3 Create new resource modules.

Compile the resource files into DLLs. Use the following commands to do this step:

c:\progra~1\borland\delphi5\bin\dcc32 rbPrint.dpr
c:\progra~1\borland\delphi5\bin\dcc32 rbIDE.dpr
c:\progra~1\borland\delphi5\bin\dcc32 rbDADE.dpr

After these commands have executed, you should have three cst files in the cst folder: rbPrint.cst, rbIDE.cst, and rbDADE.cst.

4 Install new resource modules.

The final step is to copy the new .res files and DLLs to the ..RBuilder\Lib and Windows\System directories respectively. Use the following commands to do this step:

```
copy rbPrint.cst c:\windows\system
copy rbIDE.cst c:\windows\system
copy rbDADE.cst c:\windows\system
```

Note: If you are using Windows NT, you will need to copy these files to Windows\System32.

5 Use the custom translation.

Create a report within Delphi; select the lgCustom language type, and double-click on the report to display the Report Designer. Your translation will be displayed.

REPORTBUILDER ENTERPRISE EDITION FUNDAMENTALS

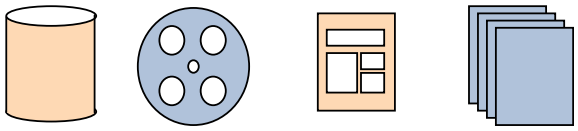
Main

MAIN

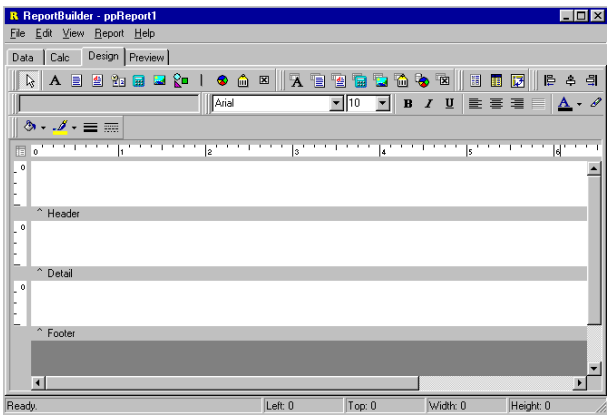
Introduction

Overview

The ReportBuilder Enterprise Edition includes everything in ReportBuilder Standard, plus a full set of components necessary to deliver a complete end-user reporting solution. In the introduction to ReportBuilder, the reporting equation is described. The reporting equation divides reporting into four main activities:



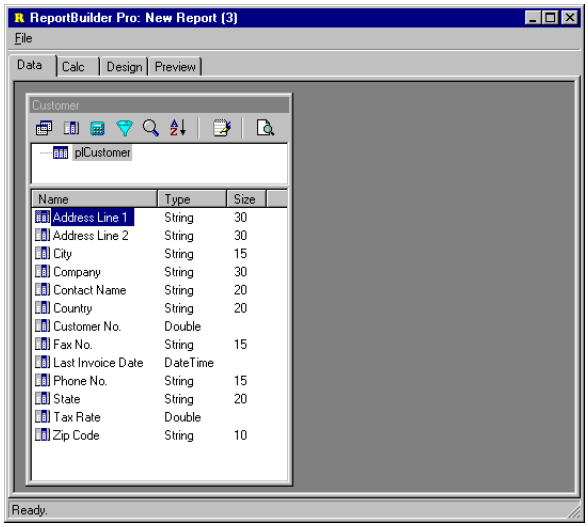
In ReportBuilder Enterprise, the goal is to deliver a full-fledged reporting solution to end users. This goal is achieved by delivering visual, easy-to-use solutions in each of these four areas. This screen shot of the ReportBuilder Enterprise Report Designer shows the ergonomic design of the user-interface.



Each of the four areas of reporting has a representative notebook tab containing a visual environment for the creation and configuration of components within that area. The results of each area become input for the next area: data feeds into calculations, calculations feed into components within the report layout, and the report layout is rendered into a preview of the report. The implementation used by ReportBuilder Enterprise for each area of reporting is described below.

Data

Within the work environment of the Data tab, end users can quickly create dataviews, which can then be used to supply data to reports. Dataviews are usually created via the Query Wizard or Query Designer. Both of these tools are visual; they also allow the end user to select the tables, fields, search criteria, and sort order necessary for the report. Behind the scenes, an SQL statement is generated and used to retrieve the data from the database. A screen shot of a completed dataview is shown below.



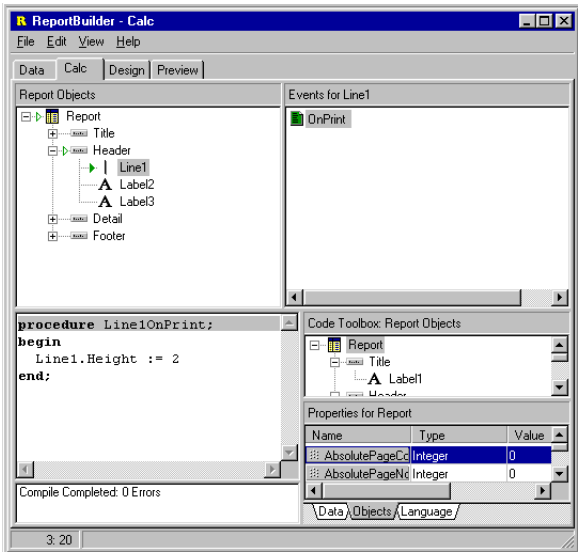
The solution described on the previous page is the standard behavior within the data workspace. However, the developer can customize this user-interface by performing one of these three tasks:

- Register a replacement query wizard or query designer.
- Remove the query wizard or query designer.
- Create new dataview template classes that can simplify the data selection process even further by establishing the relationship between the tables in the database and presenting an alternative user-interface (such as a single form that allows search/sort criteria to be entered).

The bottom line is that the Data area contains a turnkey solution that can be used out-of-the-box, but if customizations are needed, an architecture has been provided so that those customizations are possible.

Calc

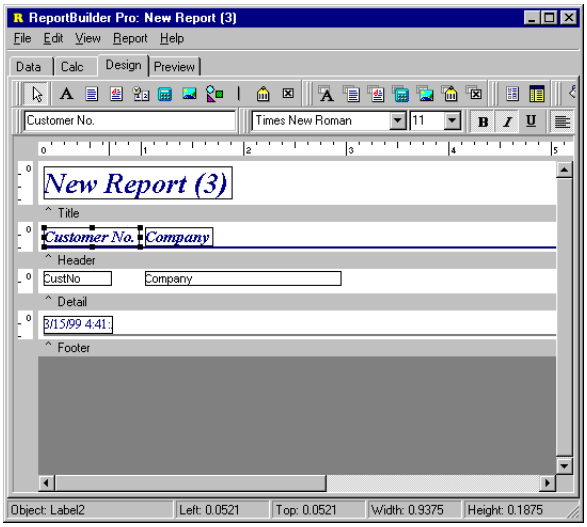
This workspace contains a tree view of the report, all the bands within the report, and all the objects within each band. When a band or component is selected, all the events for that component are shown in a list. The user can then select an event and code the event handler in the syntax-sensitive code editor at the bottom. The following screen shot shows an OnPrint event as coded for a Line component.



This screen shot shows the Calc workspace in its most feature-rich and complex configuration. Development work completed here can be passed on to end users so that they can modify it, locked down so that end users can only view it, or hidden completely so that end users do not know it is there. The most scaled-down version of the Calc tab is provided by a dialog-only interface, where no Calc tab exists at all, and a single syntax-sensitive code-editor dialog is accessible from the Calculations... menu option of the variable component. Again, the user-interface and behind-the-scenes architecture has been made highly scalable in order to meet the various needs of developers.

Design

The Design workspace contains the actual layout of the report. The user-interface is identical to the one presented to developers using ReportBuilder at Delphi design-time; in other words, it is full-featured and professional. The Office97-style interface makes the Design workspace especially easy to learn for end users. A Report Wizard is available for creating reports quickly. You can customize this interface by replacing any of the dialogs it uses and by registering your own report wizards.



Preview

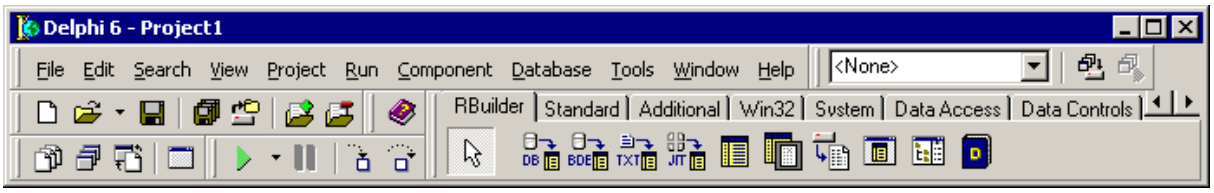
The Preview workspace contains the rendered report. The report can be printed to the printer or to various file formats from this workspace.

The Delphi Components

Overview

The following components appear on the RBuilder tab of the Delphi component palette when you install ReportBuilder Enterprise

Delphi Components



DBPipeline

Used for accessing data via the BDE, third-party BDE replacement products, or TDataSet descendants. The DBPipeline is connected via the DataSource property.

BDEPipeline

In previous versions of ReportBuilder, the BDEPipeline was used for accessing data via the BDE. Though it has been replaced by the DBPipeline, it has been retained for backward capability.

TextPipeline

Used to access comma, tab, and fixed-length record text files. Set the FileName property to specify the file. Double-click on the component to define the field structure.

JITPipeline

Used to access any non-structured data stored in Delphi objects or other sources. Provides total Set the InitialIndex and RecordCount properties and code the OnGetFieldValue event to utilize this component. Double-click on the component to define the field structure.

Report

The main component. Double-click to invoke the Report Designer. Assign the DataPipeline property so that the report can traverse data. Assign the DeviceType property to control where the output of the report is directed. Call Report.Print from Object Pascal to print the report or launch the Print Preview Form.

Viewer

This object is rarely used because you can replace ReportBuilder's built-in print preview form with your own customized version very easily (check the Building a Reporting Application tutorial). If you must use this component, an example is provided in \RBuilder\Demos\Reports.

ArchiveReader

After you print a report to an archive file (.raf extension), you can read and preview the file via this component. Just assign the ArchiveFileName to the file and call the Print method. In terms of displaying a report, this component works the same as the Report component.

DataDictionary

Used by the QueryWizard component to convert raw table and field names into more usable aliases. This component is generally used only when you are creating an end-user reporting solution.

Designer

Used when you want to deploy the Report Designer to your end users. See the End-User demo (Demos\EndUser directory).

ReportExplorer

Used to provide a Windows Explorer-style interface to End-User Reporting solutions developed using ReportBuilder Enterprise. See the End-User demo (Demos\EndUser\1 Report Explorer\ directory).

Designer Component

Overview

The Designer is a non-visual component that acts as a 'wrapper' around the Report Designer window displayed in the end-user reporting solution. By setting the various properties and events of the designer component, you customize the content and behavior of the Report Designer. The most commonly used properties are listed below.

AllowDataSettingsChange

Used to control whether the DataSettings menu option appears in the File menu of the data workspace. Usually disabled to prevent end users from changing the database settings.

AllowSaveToFile

When set to True, the File menu of the design workspace has 'Load from File' and 'Save to File' menu options.

DataSettings

DADE only. Used to configure DADE (data workspace) for a given database.

Icon

The icon to be displayed on the Report Designer window.

RAPInterface

RAP Only. RAP configuration settings.

RAPOptions

RAP Only. Further RAP configuration settings.

Report

The report object that will be used to load and save reports. The configuration of the Report Template properties is used by the Report Designer to determine how reports will be loaded and saved.

ShowComponents

Determines which components will be displayed in the ReportBuilder component palettes. Sometimes used to disable advanced components such as regions or subreports.

TabsVisible

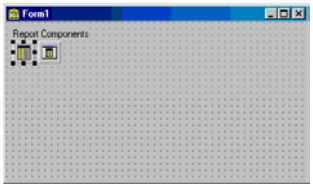
Determines whether the Design and Preview tabs will display. When set to false, the Report Designer becomes a 'design only' window, where the end user can work on the layout, but cannot preview. If DADE or RAP is installed, then this property cannot be set to False.

A Simple End-User Report Solution

We can create a simple end-user reporting solution using just a designer component and a report component.

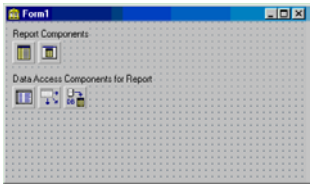
1 Create Report and Designer components.

Place a designer component and a report component on a form, then connect the report to the designer by assigning the Designer.Report property.

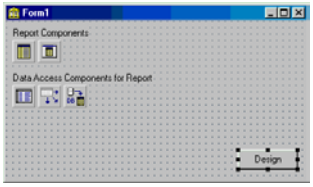


2 Create data access components.

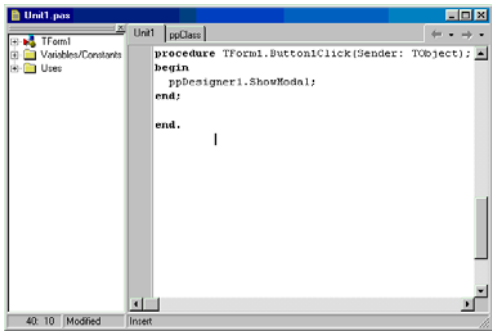
Create data access components to supply data to the report. Here we have created TTable, TdataSource, and TDBPipeline components. The TableName of the TTable component has been set to 'customer.db' so that the user can create a report based on the data in this table.



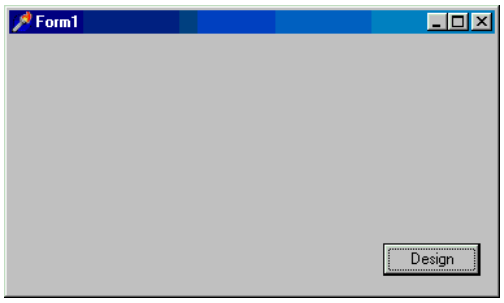
3 Create a button to launch the Report Designer.



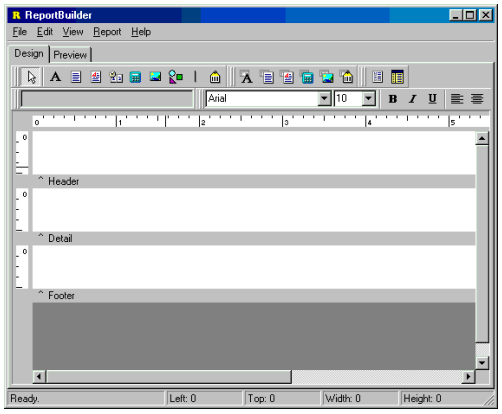
4 Code the OnClick event of the button to launch the designer.



5 Run the application.



6 Click the 'Design' button to launch the designer.



The user can now design a report based on the customer table. The report layout can be saved to a file, or pre-existing layouts can be loaded from file. Reports can only be created based on the customer table, since that is the only table available to the report.

While this end-user solution is simple and easy, it does have some limitations:

1 This approach does not provide a data workspace where different data besides the customer table can be accessed.

2 This approach does not provide a calc workspace where custom calculations or event handlers can be assigned to the report.

3 The ability to save or open report layouts to file is dialog-based, and therefore the user does not have a way to organize and keep track of the many reports he may create.

These three limitations can be addressed with additional functionality available in ReportBuilder Enterprise, namely DADE, RAP, and the Report Explorer. These solutions are discussed at length throughout this section of the guide.

Report Explorer

Overview

The Report Explorer allows your end users to use a Windows Explorer user-interface to manage reports stored in a database. The Report Explorer is a non-visual component that acts as a 'wrapper' around the explorer form, which is displayed at run-time. You can set the value of various properties at design-time via this component, which will then affect the behavior of the report explorer form at run-time. The most commonly used properties are listed below.

Designer

The designer component that will be used to display the Report Designer window when report layouts are created or edited.

FolderFieldNames

The names of the fields to be used when storing a folder.

FolderPipeline

The data pipeline to be used when storing a folder.

ItemFieldNames

The names of the fields to be used when storing an item (i.e. report, code module or data module).

ItemPipeline

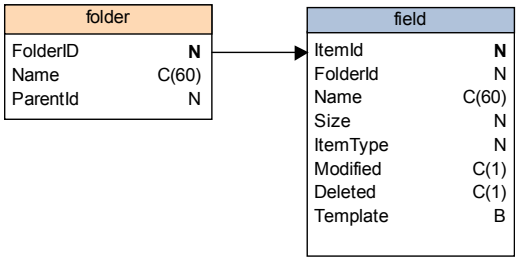
The data pipeline to be used when storing an item.

ModalForm

Whether or not the report explorer form should be displayed modally. Defaults to True.

The report explorer is designed around the concept of folders and items. 'Items' include report layouts, data modules, or code modules. Report layouts consist of everything the end user sees when designing a report, including the dataviews in the data workspace and the event handlers coded in the calc workspace. Data modules consist of dataviews that have been exported from the data workspace so that they can be imported for use in creating new reports. Code modules consist of event handlers that have been exported from the calc workspace so that they can be imported into other reports.

The items displayed in the explorer must belong to a certain folder. Folders may be nested within other folders, or they may be assigned to the global parent folder (entitled 'All Folders'), which is always displayed. Both the folders and items are stored via data pipelines.



The FolderId field of the Folder table is the key field and must always be unique. In Paradox, this field is usually an AutoIncrement. In SQL database tables, this field is usually a sequence number generated by a trigger. Both techniques are documented in the end-user tutorials. The relationship between the folder and item table is one to many. This is not to say that the item table does not use the Name field as a key: it uses an ItemId field. This field is also an auto-generating sequence number. While it is usually true that two items with the same name cannot reside in the same folder (same name and folder id), this is not true for the recycle bin, which can contain multiple items with the same name. Thus, the FolderId-ItemType-Name fields cannot form a unique key (the item type field indicates whether the item is a report layout, data module, or code module).

All items assigned to the recycle bin have a folder id of -2. All items assigned to the global folder 'All Folders' have a folder id of -1. Items assigned to any other folder have that folder's id value in the folder id field. Neither the 'All Folders' folder nor the recycle bin actually appear in the folders table: they are automatically created by the report explorer.

Data Dictionary

Overview

The data dictionary is a non-visual component designed to provide the data workspace with the capability to replace the table names and field names retrieved from the database with more readable aliases that you provide. The most commonly used properties of the data dictionary are described below.

FieldFieldNames

The names of the fields to be used in accessing the field aliases.

FieldPipeline

The name of the data pipeline that has access to the field aliases.

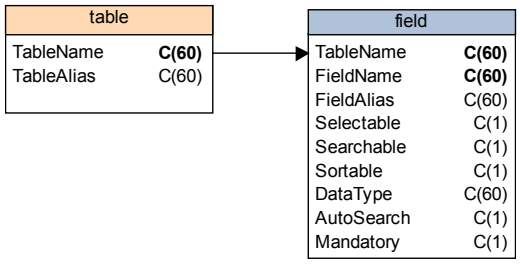
TableFieldNames

The names of the fields to be used in accessing the table aliases.

TablePipeline

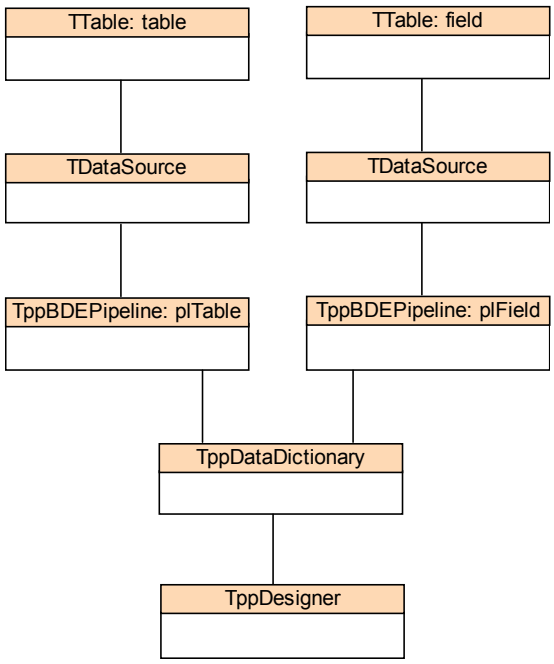
The name of the data pipeline that has access to the table aliases.

The data module for the database tables underlying the data dictionary is shown below.



The TableName is the key field of the table table. The TableName and FieldName are the key fields of the field table. The Selectable field determines whether the field can be selected within the Query Wizard and Query Designer. The Searchable field determines whether the field can be used in search criteria. The Sortable field determines whether the field can be used to order the data.

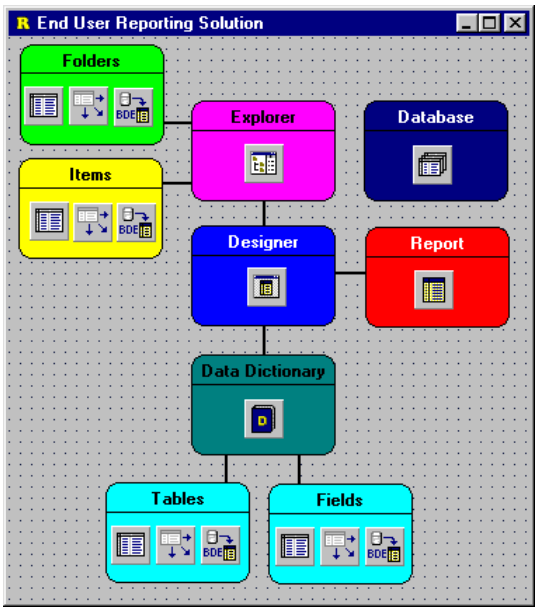
Once the data dictionary tables have been created and populated, they can be assigned to a standard Delphi TTable or TQuery component, which can then be assigned to a data pipeline via a TDataSource. The pipelines are then assigned to the TablePipeline and FieldPipeline properties of the data dictionary. The diagram below shows a fully configured data dictionary component. The data dictionary component is assigned to the data workspace via the DataDictionary property of the designer component.



Putting It All Together

Summary

In order to create the most sophisticated and complete end-user reporting solution possible, we need to connect and configure all of the components available in ReportBuilder Enterprise. The screenshot below shows the configuration of a full-featured end-user reporting solution (the colored shapes and labels have been added for informational purposes only). A completed version of this solution is available in the \RBuilder \Demos\EndUser\1. Report Explorer directory. Step-by-step instructions for creating this application are provided in the 'Building an End-User Reporting Application' tutorial.



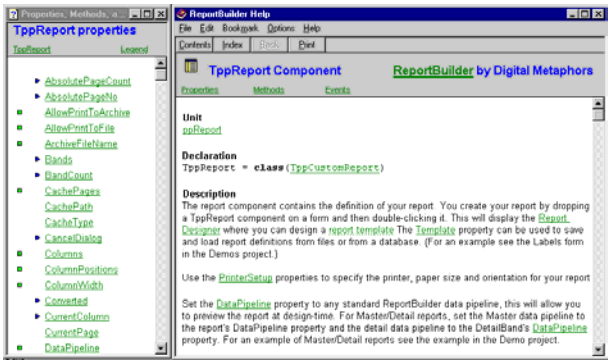
On-Line Help

Overview

The ReportBuilder On-Line Help contains over 1,500 topics documenting the properties, events, and methods of the various components that make up the product.

For maximum readability, the help follows the exact style of the Delphi help. It is designed to be integrated with the Delphi help so that when you access the Delphi help (via the Help | Topics menu option), 'ReportBuilder Reference' appears as a single entry on the contents page. Topics from the ReportBuilder help will appear when you search the help from the 'Index' page. Also, when you create a component on a Delphi form, you can access the main help topic for that component by clicking F1. If you select a component property in the Object Inspector and click F1, the help for that individual property will be displayed.

The help is designed as a reference for those who have already mastered the basics of ReportBuilder. In order to get started using the product, study this guide and complete the tutorials.



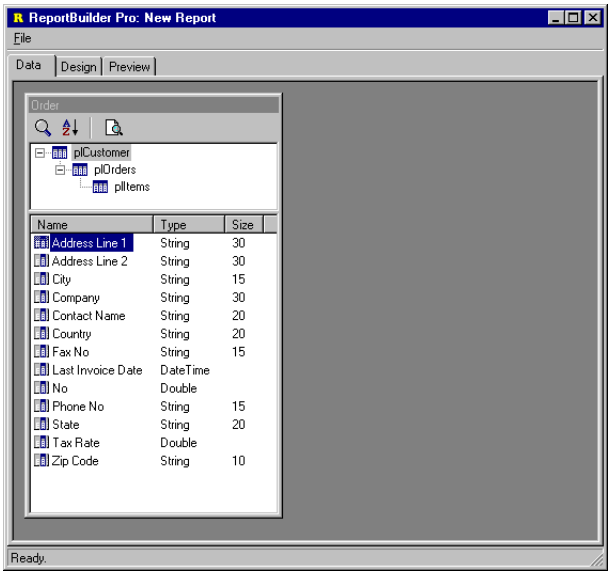
DATA

DATA

Introduction

Overview

The developers of ReportBuilder discovered that a visual solution for data access was needed for end users. This realization led to the development of DADE, the Data Access Development Environment. Although DADE consists of an extensive object-oriented architecture 'under the covers', it appears as a simple, easy-to-use data workspace within the Report Designer. The data workspace of the Report Designer, which contains a completed dataview, is pictured below.



Dataview

A dataview presents what appears to be a single set of data to the end user. In reality, the dataview may be composed of one or many Delphi data access components. The implementation of a dataview can be as simple as a single SQL query, or as complex as a set of linked tables. Whatever the implementation, the dataview always appears the same to the end user, and the implementation always depends upon the choices you make as a developer.

Dataviews interface with the other workspaces within the Report Designer via the data pipeline(s) they contain. The dataview to the left contains a set of master/detail tables. These tables feed data through standard Delphi TDatasource components, which in turn feed the data through data pipeline components. When this dataview is first created, it assigns the customer data pipeline to the report. The order and item pipelines are available to the end user for use in subreports.

The data pipeline components within a dataview create the interface between the dataview and the other workspaces within the Report Designer. Within the design workspace, the data pipelines for all dataviews are listed in the data tree, the drop-down list of the edit toolbar, and in the data dialog, which is accessible from the main menu. Within the calc workspace, the data pipelines appear in the object list and in the code toolbox, making it easier to code calculations involving field values.

We will discuss DADE, dataviews, and how to use this part of ReportBuilder to maximize the productivity of your end users in the following section.

Query Wizard

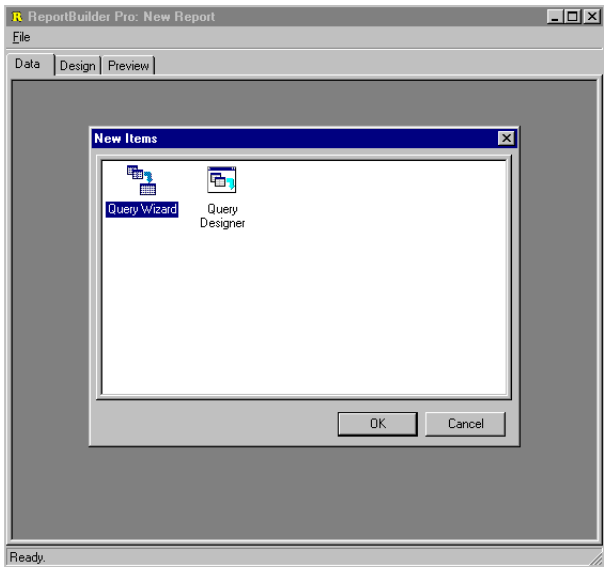
Overview

Within the data workspace, you can select data from your database using an SQL query. This functionality is provided via query-based dataviews, which can be visually created using the Query Wizard and visually maintained using the Query Designer.

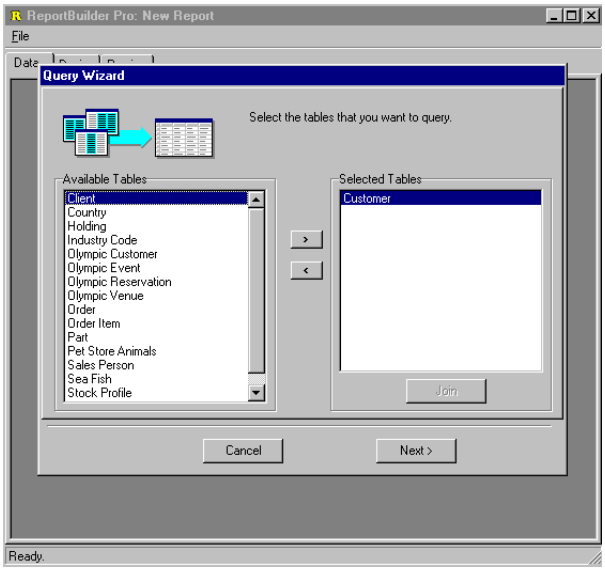
Query Wizard: Create a Simple Query-Based Dataview

The following series of screenshots shows how to create a simple query-based dataview via the Query Wizard.

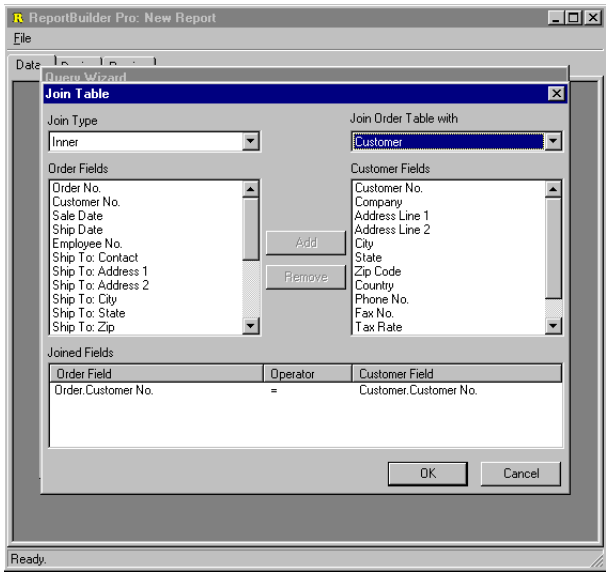
1 Select File | New from within the data workspace. The New Dialog will be displayed. Double-click the Query Wizard icon.



2 Select the first table for the query. The customer table has been selected.

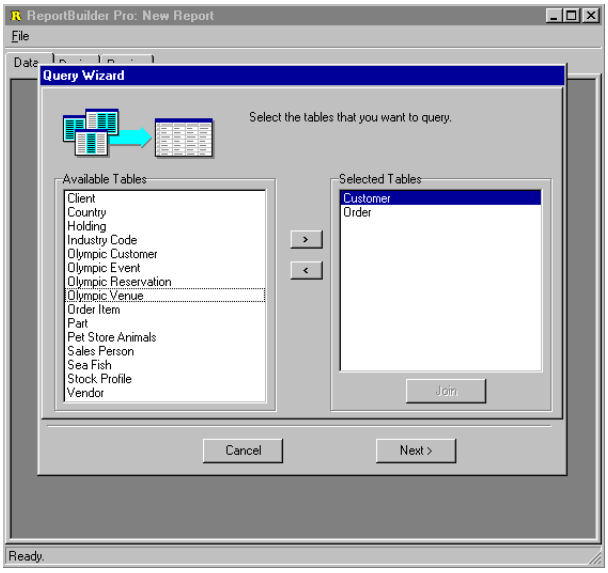


3 Select the second table for the query. The order table has been selected and the Join dialog was automatically displayed. The dialog already contains the correct field linking (Customer No.), so all we have to do is click OK.

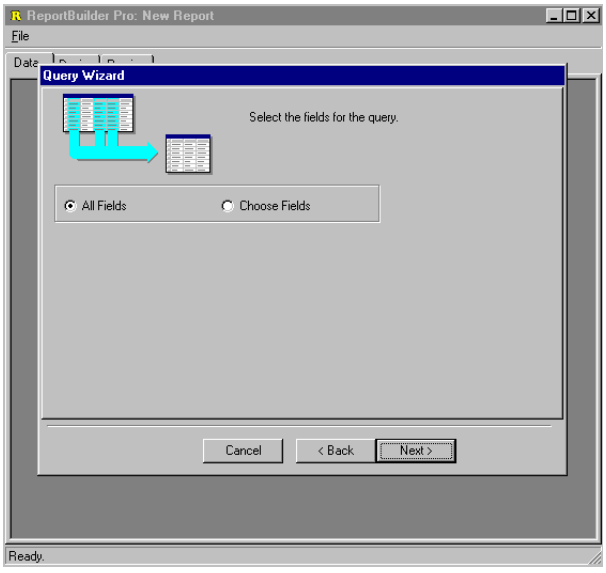


Query Wizard: Create a Simple Query-Based Dataview - cont.

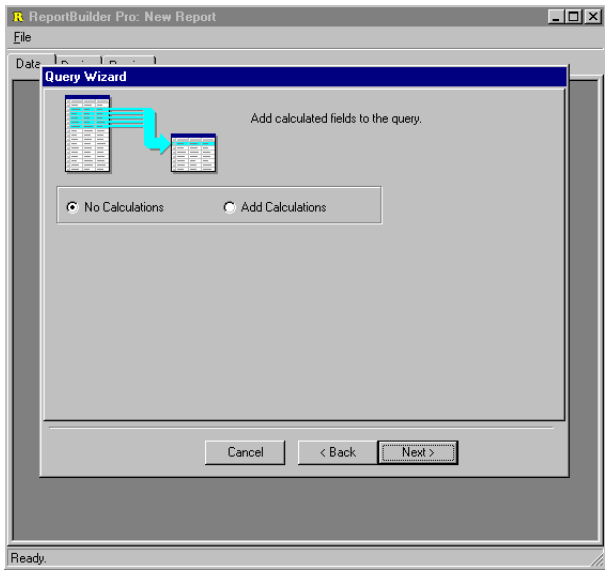
4 When we return to the query wizard, both tables are shown as selected.



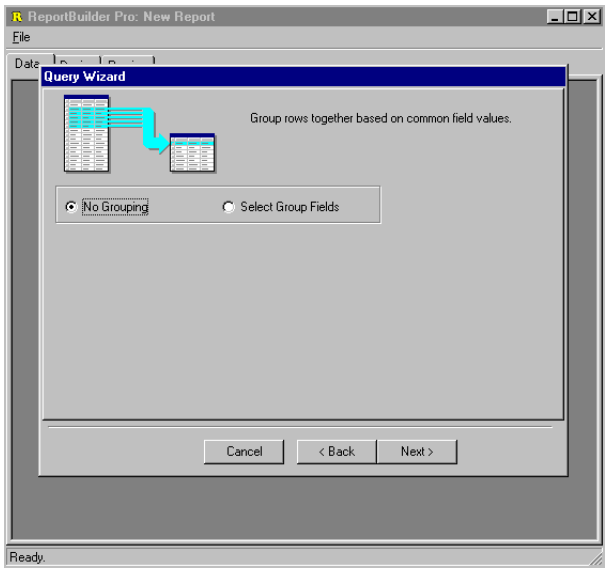
5 Skip the fields page, since we want to select all fields.



6 Skip the calculations page, since this query will not contain calculations.

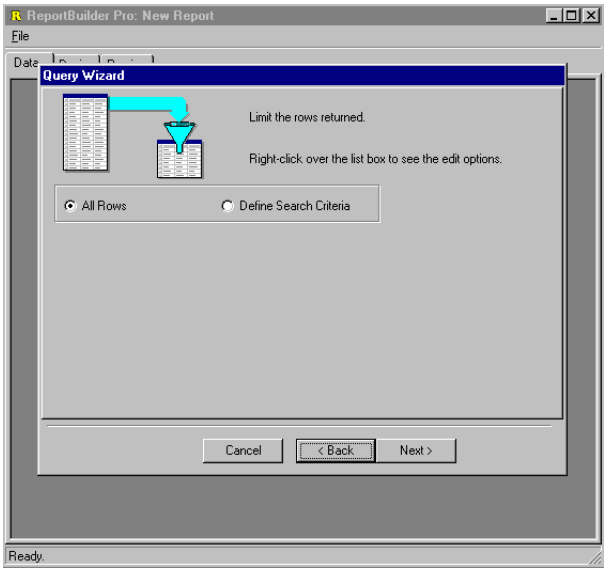


7 Skip the groups page, since this query will not be grouped.

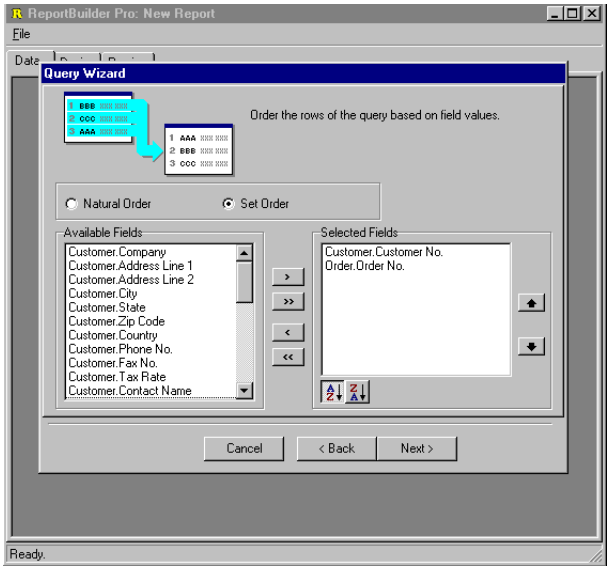


Query Wizard: Create a Simple Query-Based Dataview - cont.

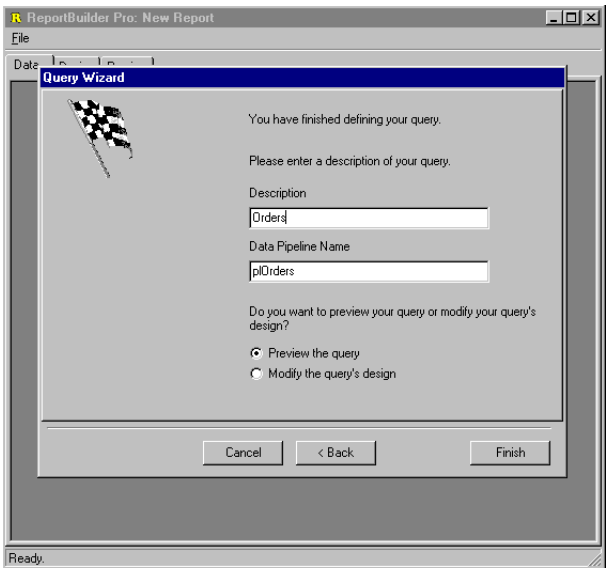
8 Skip the search criteria page; all records will be selected.



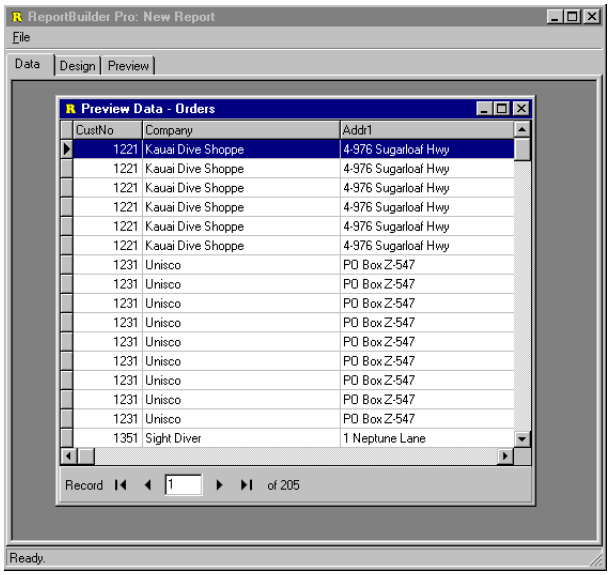
9 Set the order to customer number, then order number.



10 Name the dataview. The data pipeline name is automatically generated when we name the dataview. The next action will be to preview the data.

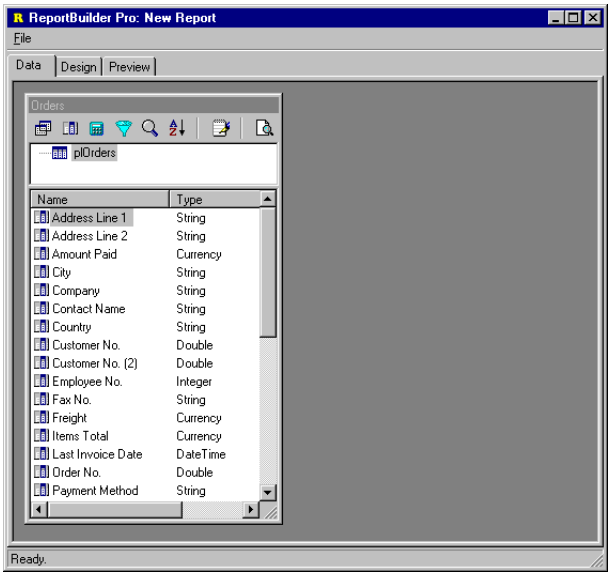


11 View the data to make sure the correct records have been selected.



Query Wizard: Create a Simple Query-Based Dataview - cont.

12 Close the preview window. The dataview is then created and displayed in the workspace.

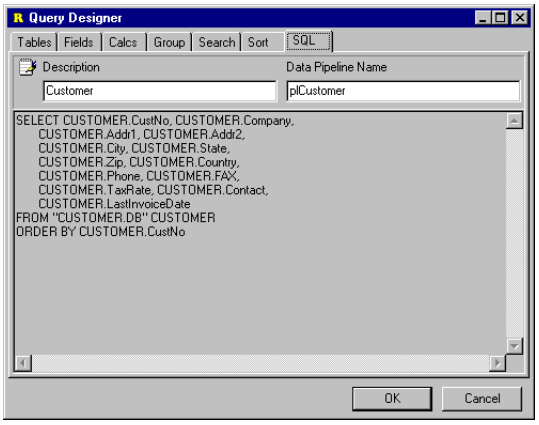


From here we could proceed to the design workspace where we could create a new report layout based on this dataview either manually or through the use of the Report Wizard.

Query Designer

Overview

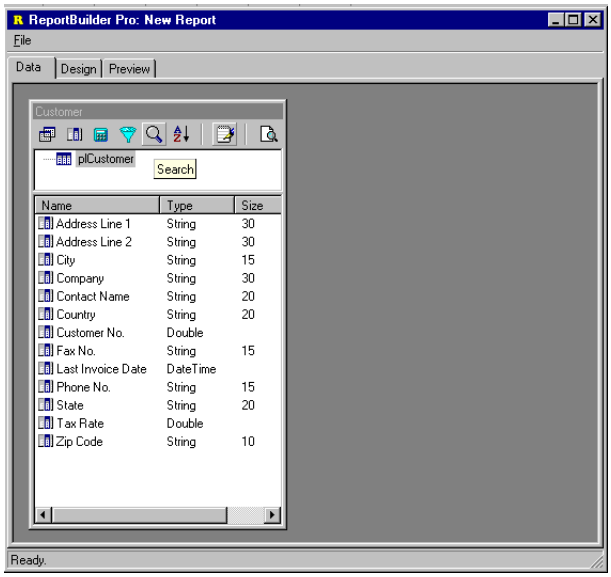
The Query Designer is used to modify query-based dataviews. The query designer presents a series of notebook tabs; each tab represents a different part of the query. The last notebook tab in the Query Designer shows the generated SQL and allows the name of the dataview and data pipeline to be changed. The Query Designer is pictured below.



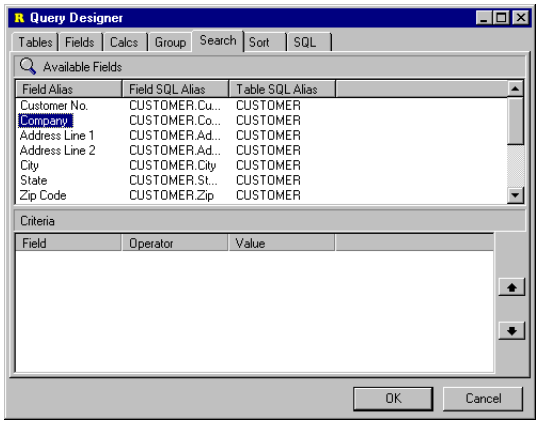
Query Designer: Adding Search Criteria

You can use the Query Designer to add or remove search criteria from your query. Perform these steps in order to add search criteria:

1 Click on the Search icon of the dataview to launch the Query Designer.

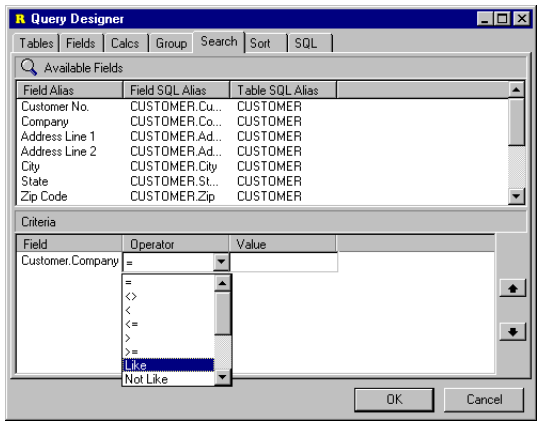


2 From the list of fields at the top of the search page, double-click on the field for which criteria needs to be entered.

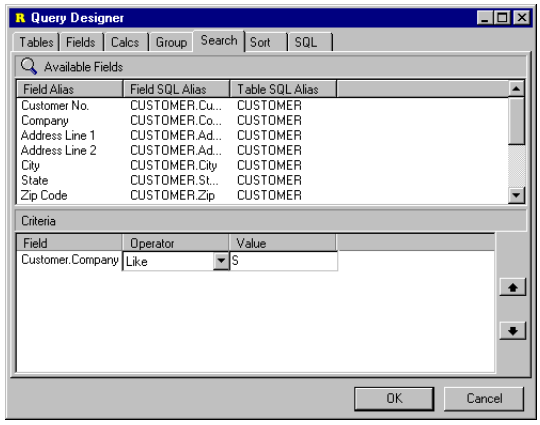


Query Designer: Adding Search Criteria - cont.

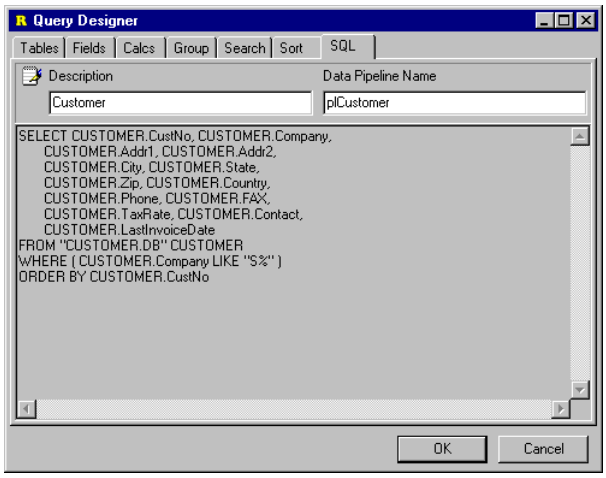
3 Click on the field that has been added to the list of criteria at the bottom and select the operator.



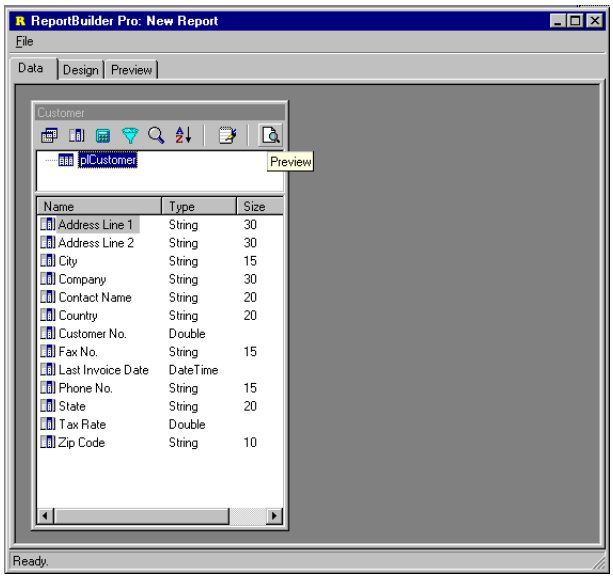
4 Click in the edit box and enter the search criteria value. This criteria will find all company names that begin with the letter 'S'.



5 Click on the SQL tab to make sure the criteria value is valid.

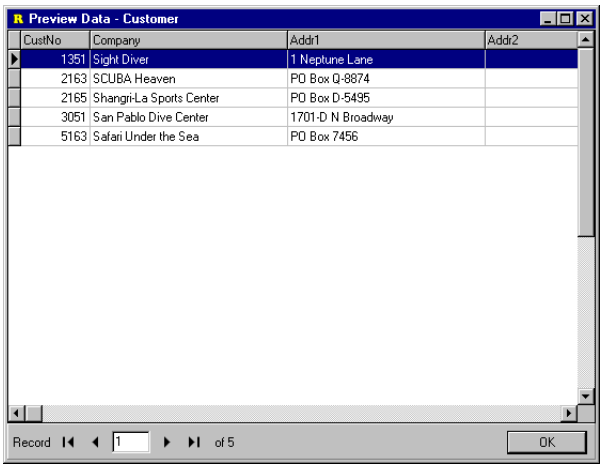


6 Close the Query Designer and click on the preview icon.



Query Designer: Adding Search Criteria - cont.

7 Preview the data and make sure that the intended records are selected.



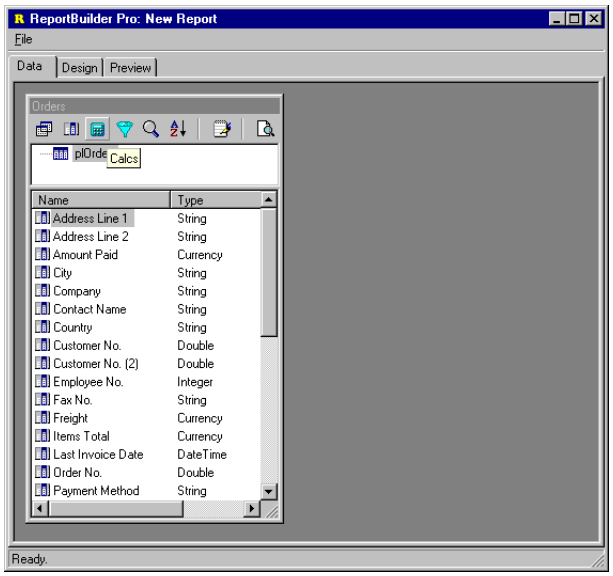
Create a Group Sum

The SQL 'GROUP BY' clause allows you to eliminate rows in your query where the field values repeat. For example, let's assume we have a database table that contains order records. Each order record has the customer number and the amount paid. If we viewed the data in this table, we would see that the value in the customer number field repeats where there are multiple orders for a customer.

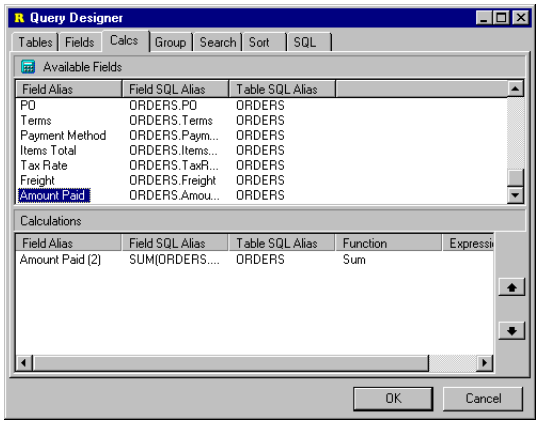
We can use SQL to select data from the orders table and calculate the total amount paid for each customer. We can do this by specifying a group on the customer number field. By specifying the group, we are saying to the SQL engine: create one row in the result set for each customer number found. When the SQL engine runs the query, it will find multiple records for some customers; these records will be eliminated from the result set. SQL allows us to perform calculations on these repeated records and store the result in a new field of the result set.

These types of calculations can be created on the Calc tab of the Query Designer. Perform these steps in order to sum the amount paid for all customers in the orders table:

1 Click the 'Calc' icon to launch the Query Designer.

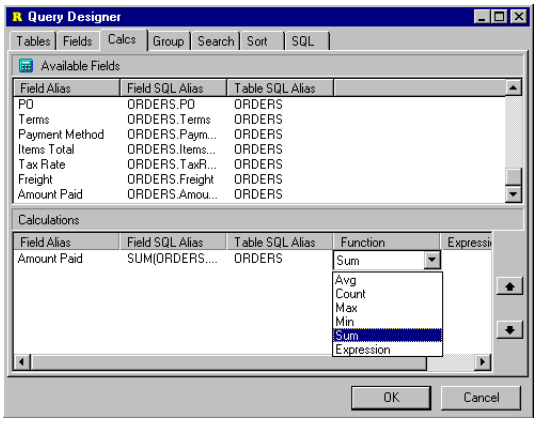


2 Double-click the 'Amount Paid' field from the selection list at the top of the page. Amount Paid will be added to the list of calculations.

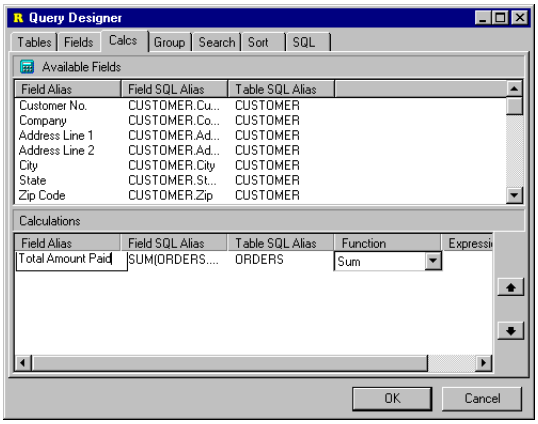


Create a Group Sum - cont.

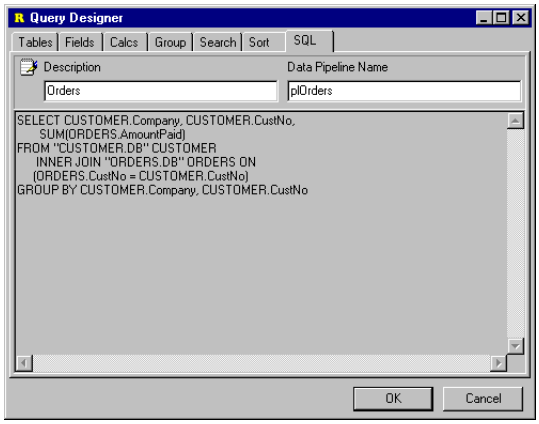
3 Select 'Sum' as the function type for the calculation.



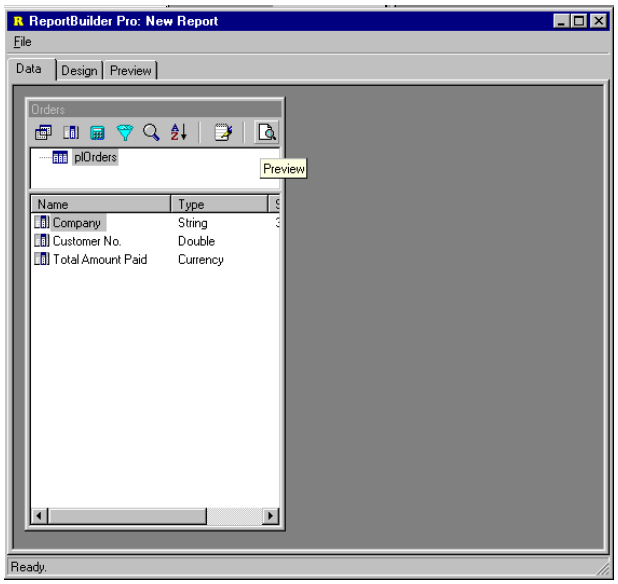
4 Enter the Field Alias you would like to use for this calculated field.



5 Click the SQL tab to make sure the generated SQL is valid.

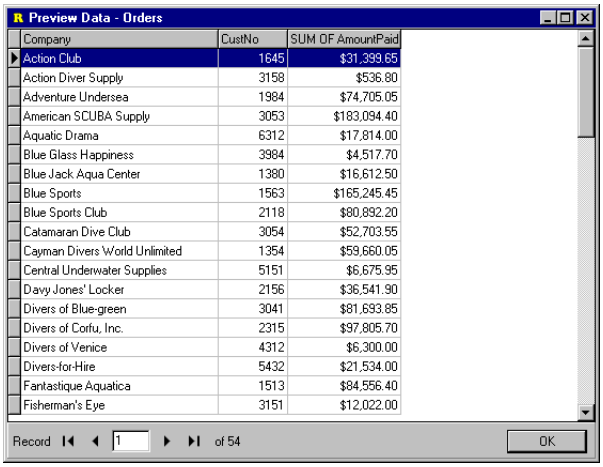


6 Close the Query Designer and click the Preview icon to preview the data.



Create a Group Sum - cont.

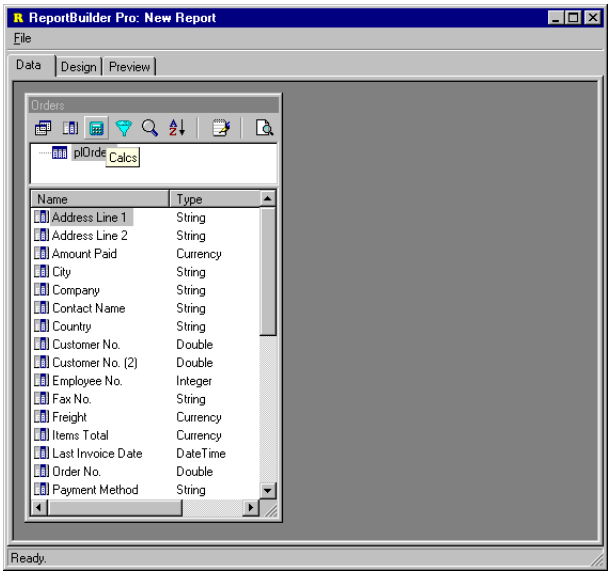
7 Check the data to make sure the sum is calculated as expected.



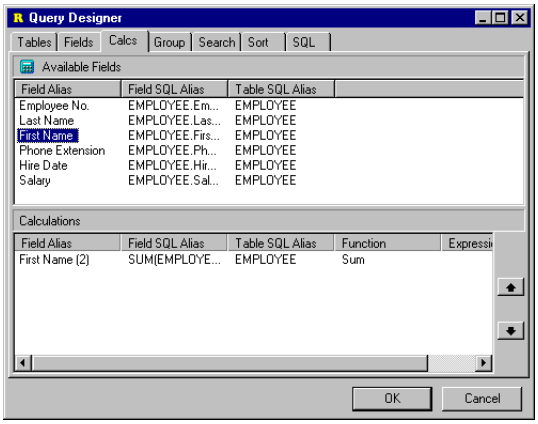
Concatenate Fields

You can enter SQL expressions from the Calc tab of the Query Designer. The following query selects data from a table of employees. The table has a first name and last name field. Perform these steps in order to concatenate these two fields together using the Query Designer:

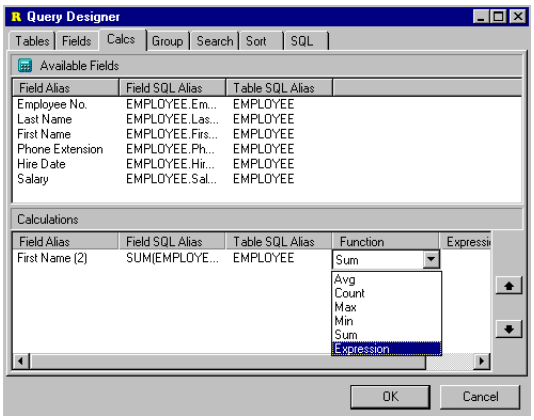
1 Click the 'Calcs' icon to launch the Query Designer.



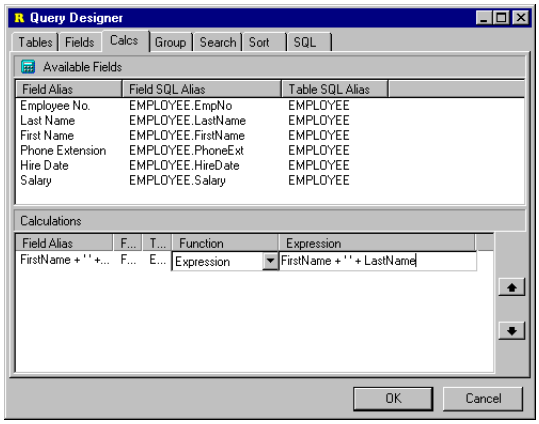
2 Double-click the 'First Name' field from the selection list at the top of the page. 'First Name' will be added to the list of calculations.



3 Select 'Expression' as the function type for the calculation.

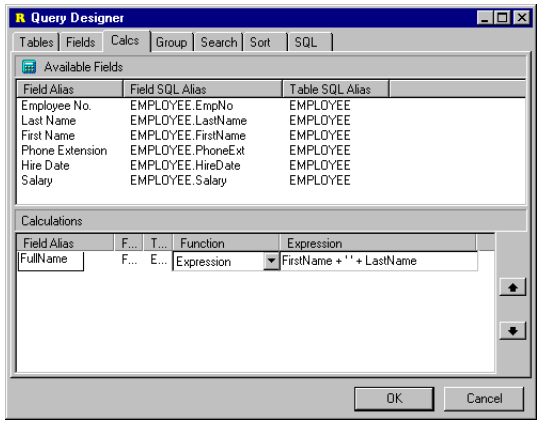


4 Modify the widths of the 'Field SQL Alias' and 'Table SQL Alias' and 'Expression' columns in the calculations list at the bottom of the page so that there is enough space to enter the expression. The figure below illustrates a sample expression.

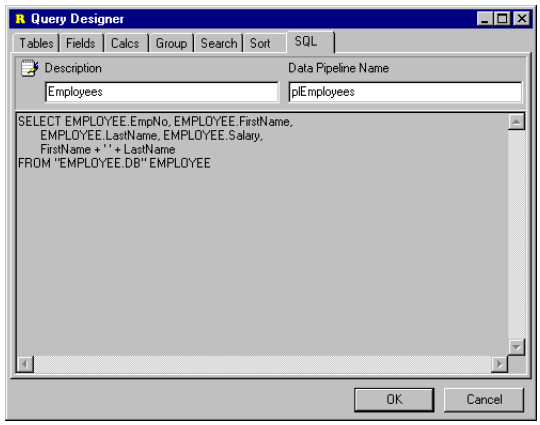


Concatenate Fields - cont.

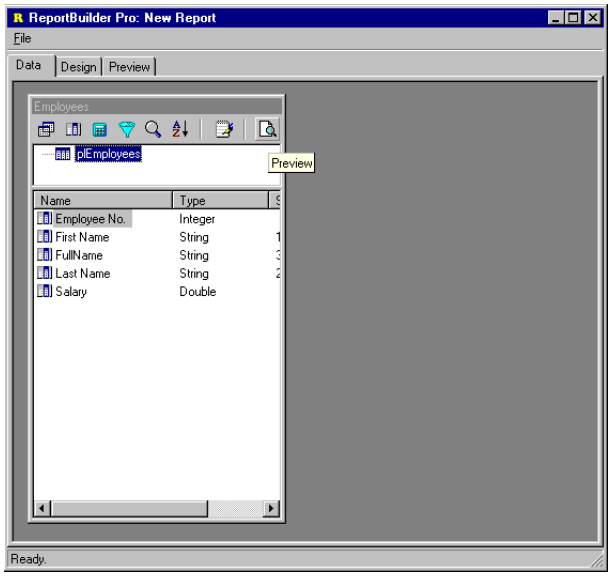
5 Enter the Field Alias you would like to use for this calculated field.



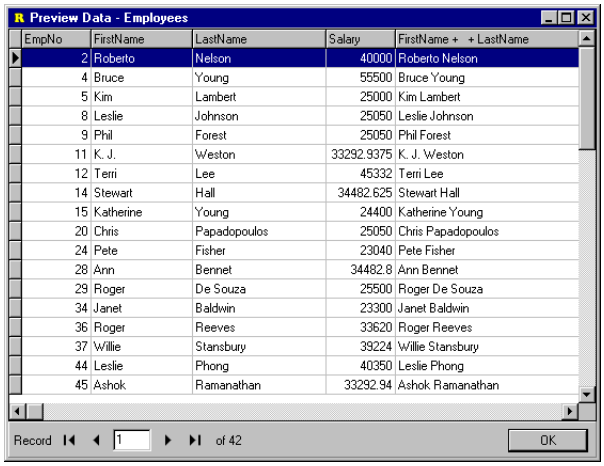
6 Click the SQL tab to make sure the generated SQL is valid.



7 Close the Query Designer and click the Preview icon to preview the data.



8 Check the data to make sure the field is calculated as expected.

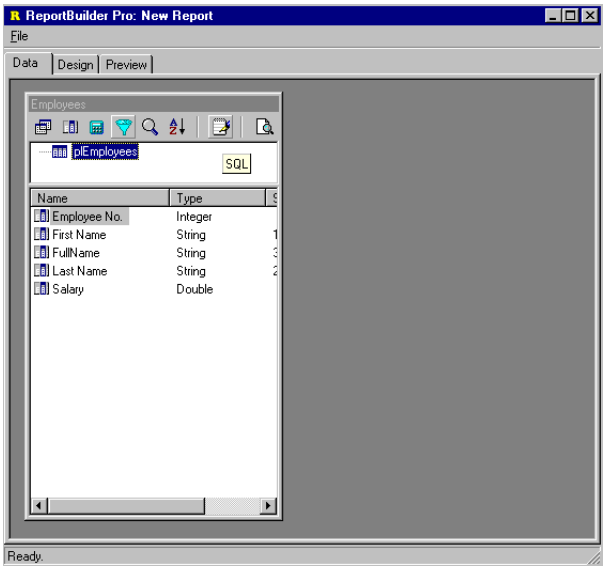


Edit SQL

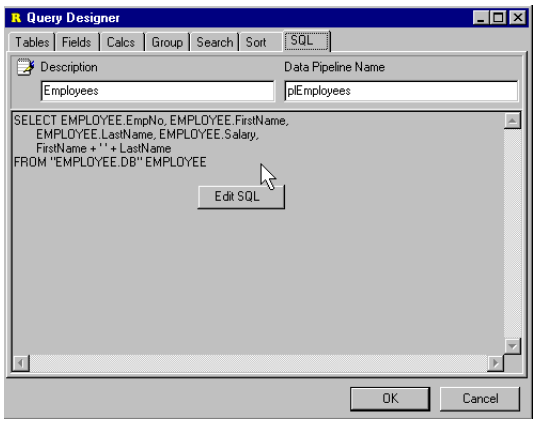
There may be times when you need to utilize advanced features of SQL that cannot be accessed via the visual interface of the Query Designer. In these cases, you can edit the SQL manually in the Query Designer. Once you have edited the SQL manually, you must always use the SQL tab of the Query Designer to make future modifications.

Perform these steps in order to edit the SQL generated by the Query Designer:

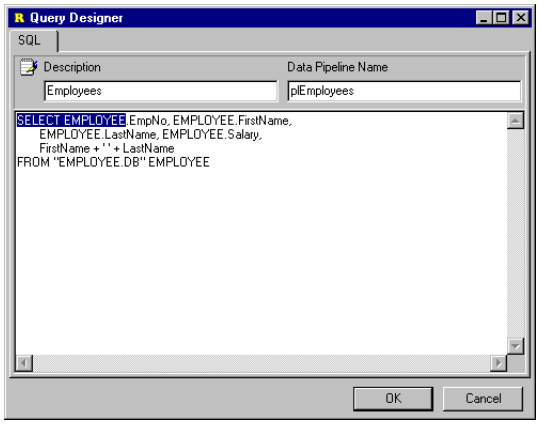
1 Click on the SQL icon to launch the Query Designer.



2 Right-click over the SQL text to display the popup-menu.



3 Select the menu item. Click Yes to the message dialog. You can now edit the SQL.



Configuring DADE

Overview
The first step in configuring DADE for your end users involves setting the properties of the DataSettings object of the Designer component. This object contains the following properties:

AllowEditSQL
Determines whether or not the end user can edit the SQL in the Query Designer.

DatabaseName
The name of the database from which data will be retrieved when the query executes.

DataDictionary
The data dictionary object that will convert raw table and field names to aliases.

SessionType
Defaults to BDESession. Should only be changed if you have created your own Session class in order to use the Query Wizard with a proprietary database engine.

SQLType
Defaults to sqBDELocal. If you are not using dBase or Paradox tables, then you should set this property to sqSQL1, as the Local SQL dialect is not supported by most other database types. If your database supports ANSI-92 SQL, then you can set this property to sqSQL2 instead of sqSQL1.

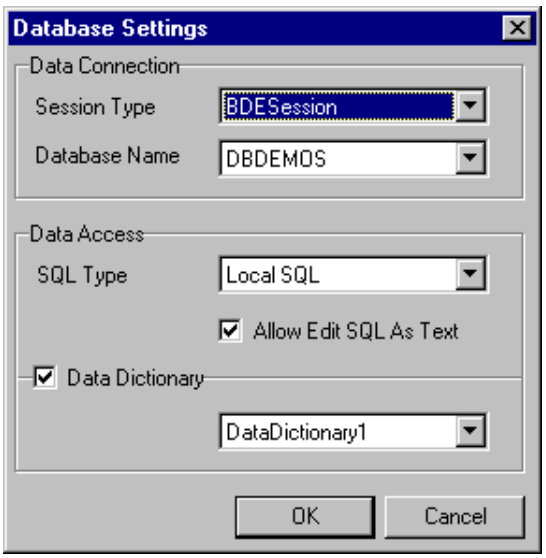
UseDataDictionary
Determines whether the data dictionary object assigned to the DataDictionary property will be used.

Database Product and SQL Type
If you intend to use the Query Wizard and Query Designer, then the SQLType is vital. The following table shows the recommended SQLType for a given database product.

Database Product	SQLType
Paradox	sqBDELocal
Interbase	SQL2
Oracle	SQL1
MS Access	SQL2
Sybase	SQL ServerSQL1
Sybase SQL	AnywhereSQL2
MS SQL	ServerSQL2

Data Settings

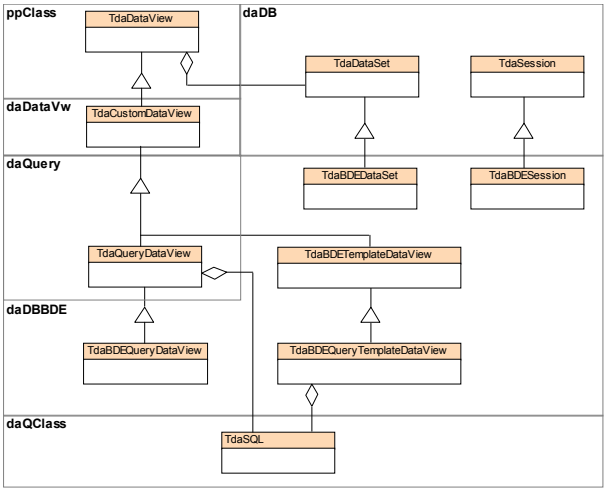
Once you have configured the data settings, you may notice that a Data Settings menu option appears on the File menu of the data workspace when you run your application. This option displays a dialog that allows the end user to modify the data settings. If you do not want the end user to be able to change these settings, then you can set the AllowDataSettingsChange property of the Designer component to False.



DADE Architecture

Overview

The Object Model containing the most important classes of DADE is pictured below:



The inheritance relationships are indicated by triangles in the above diagram. Where one class is associated with another class, a line is used. If the line terminates in a diamond shape, then it indicates that the class actually forms part of the internal implementation of the component. For instance, the TdaSQL class is part of the implementation of the TdaQueryDataView. The unit in which each class is declared is indicated by the text at the upper left corner of each containing shape.

Basic Classes

The following classes represent the heart of DADE. By creating descendants of these classes, native support for any database product that supports SQL can be implemented.

TdaDataView

Dataviews represent a set of data and are visually created and maintained in the data workspace of the Report Designer. Internally, a dataview creates the data access objects necessary to connect a report to data. The interface from the dataview into the other workspaces of the Report Designer is composed of the data pipelines, which are contained by the dataview. A dataview must have at least one data pipeline. A typical dataview will contain one or more data pipelines, datasources, and datasets.

TdaDataView contains the declarations of the methods that descendants must implement. It is in the ppClass unit because it declares a Report property that refers to TppCustomReport. If you need to know the main report with which a dataview is associated, you can always use this property.

TdaSession
Contains the methods needed by the TdaSQL class when editing (via the Query Wizard and Query Designer) and generating SQL. Provides the table names for a given database name and the database driver name (i.e. 'Sybase').

TdaDataSet
Contains methods needed by the TdaSQL class for generating SQL. It is used to get the field names for a given table name and to check SQL statements for validity.

TdaSQL
This is the main SQL editing and generation class. The Query Wizard, Query Designer, and DataViews rely on this component. As the SQL is designed visually, the SQL component maintains an object-based description of the SQL query. This description is used to generate the SQL. The SQL object is saved as part of query-based dataviews.

TdaCustomDataView
This class further defines the dataview API and implements the capability to save the dataview as a set of objects.

TdaQueryDataView
This class contains the association to the TdaSQL class.

Implementation classes

The following classes are used to implement BDE support within DADE. Non-BDE database products that replace the BDE can also be used with these classes.

TdaBDESession
Passes the method calls of the TdaSession class through to the Delphi Session object.

TdaBDEDataSet
Passes the method calls of the TdaDataSet class through to standard Delphi TTable and TQuery components.

TdaBDEQueryDataView
This class stores the results of work completed in the Query Wizard and Query Designer via the SQL object that it contains. The SQL generated by the SQL object is assigned to a standard Delphi TQuery in this class.

TdaBDETemplateDataView
This class can be used when custom dataview templates are needed. Dataview templates utilize a developer-supplied user-interface for the creation and modification of the dataview. This is generally done to replace the Query Wizard and Query Designer with a totally customized user-interface.

TdaBDEQueryTemplateDataView
This class can be used when custom dataview templates that contain a query are needed.

Extending DADE

Overview

DADE is a flexible and extensible architecture that enables developers to tailor the functionality delivered to meet the requirements of a particular application. The most common ways in which DADE is extended are as follows:

1 Dataview Templates
Create dataview templates to customize the dataview creation process or to provide data pipeline configurations that cannot be created via the built-in query-based dataview.

2 Support for Non-BDE Database
Create native support for a non-BDE database so that the Query Wizard and Query Designer can be used without the BDE.

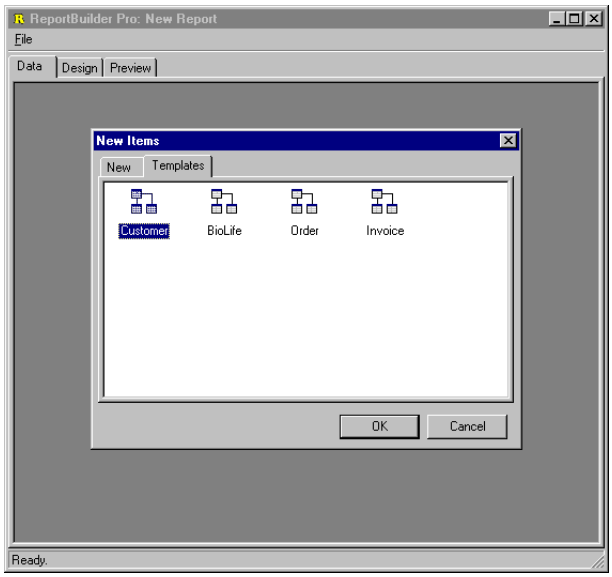
3 New Data Wizard
Create a new data wizard.

Dataview Templates

A dataview template is a special type of dataview class that has been declared by you, the developer. And while the visual representation of the dataview in the data workspace remains the same as any dataview, the user-interface used to create and modify the dataview is totally dependant on what you want to provide to end users. Dataview templates can be used for anything from creating predefined relations between a set of tables to integrating a full-blown query builder. In this section, we will show how you can create a dataview template. But before we do, let's take a look at what the end user will see when she uses a dataview template.

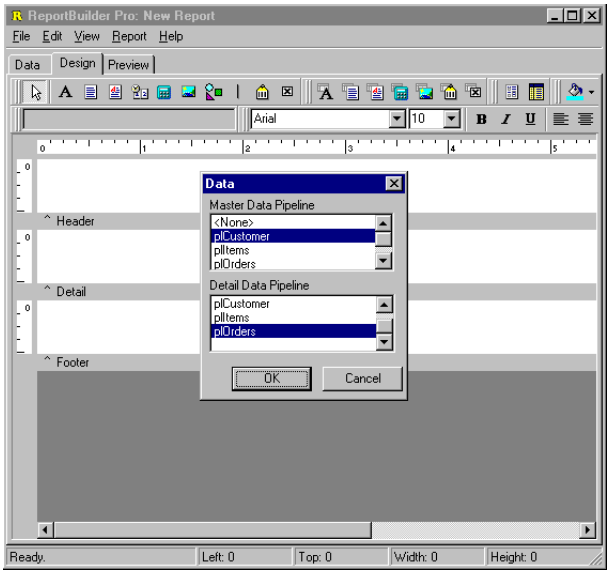
Dataview Template: The End User View

1 When dataview templates are registered with DADE, an additional tab is added to the New Items dialog. The figure below shows what we would see if we accessed the File | New menu option within the data workspace:

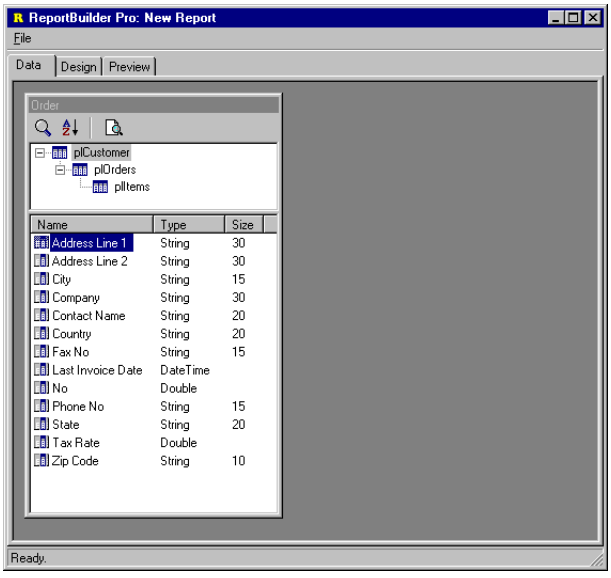


Dataview Template: The End User View - cont.

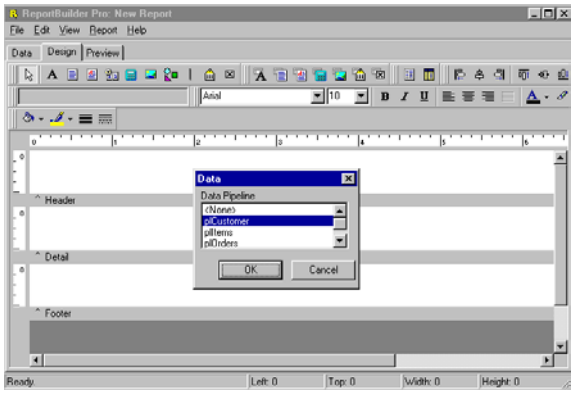
2 Here we can see that four different dataview templates have been registered. After double-clicking on the 'Order' template, the following screen would appear:



3 This is a custom dialog created by the developer of the Order dataview template. It allows the end user to enter simple search and sort criteria. Once we've done this and clicked OK, the completed dataview will be displayed:



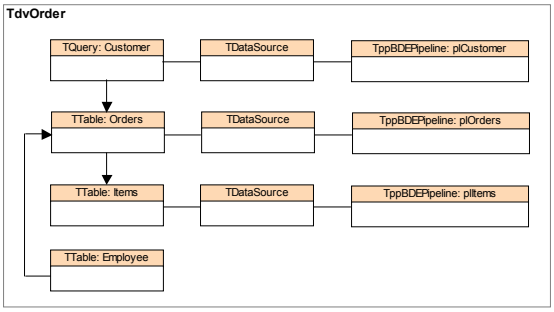
4 As you can see in the data pipeline list of this dataview, this template has declared relationships between the customer, order, and order items tables. Each table is independent and has its own data pipeline. By clicking on the Design tab and accessing the Report | Data menu option, we can see how this dataview has assigned the data pipelines to the report:



The customer data pipeline is assigned to the report. The order and item data pipelines are available, and we could create subreports and assign them to these data pipelines. This sort of data pipeline configuration is not available from query-based dataviews, which is why the developer chose to create this dataview template.

Dataview Template: The Implementation

This dataview template is implemented as an SQL query on the customer table with nested detail tables on the order and items tables. Some of the fields from the order detail table are actually supplied from the employee table, which is configured as a lookup table:



A completed and working version of the order dataview template is available in the \Demos\EndUser\2. Custom DataView Templates\ directory within the main ReportBuilder directory.

Support for Non-BDE Database

DADE is built upon the concept of the dataview; therefore, it is important to know the following traits of the dataview:

- A dataview is responsible for creating the data access objects required to connect a report to data.
- A dataview must contain at least one data pipeline.
- A typical dataview will contain one or more data pipelines, datasources, and datasets.
- Dataviews are created and maintained visually in the data workspace of the report designer.

By implementing new descendants of several classes within DADE, we can integrate a non-BDE database into DADE so that the visual tools provided in the data workspace (i.e. the Query Wizard, Query Designer, and custom dataview user-interfaces) can be used with that database.

In the course of normal operation, the Query Wizard and Query Designer require certain information from the database, such as a list of tables from which data can be selected, or a list of fields for a given table. These visual tools never directly reference any database objects in obtaining this information, but instead rely on functionality provided by the dataview. This architecture enables these tools to be used without the BDE.

Classes

Implementing a QueryDataView descendant requires us to define the following set of classes:

TdaChildADOQuery

- Descendant of TADOQuery that can be a child of a DataView
- Override the HasParent method of Tcomponent to return True
- Must be registered with the Delphi IDE via RegisterNoIcon

TdaChildADOTable

- Descendant of TADOTable that can be a child of a DataView
- Override the HasParent method of Tcomponent to return True
- Must be registered with the Delphi IDE via RegisterNoIcon

TdaADOSession

- Descendant of TppSession
- Implements GetDatabaseNames
- GetTableNames, etc.

TdaADODataSet

- Descendant of TppDataSet
- Implements GetFieldNames for SQL

TdaADOQueryDataView

- Descendant of TppQueryDataView
- Uses the above classes to create the required Query DataSource Pipeline Report connection
- Uses the TdaSQL object built by the QueryWizard to assign SQL to the ADOQuery, etc.

DADE Plug-ins:

If you have Delphi Enterprise, you can see these classes in action by installing the ADO demo. The following DADE Plug-ins are provided with ReportBuilder Enterprise. Examples can be found in \RBuilder\Demos\EndUser Databases.

- ADO
- IBX
- IBO
- Advantage
- DBISAM

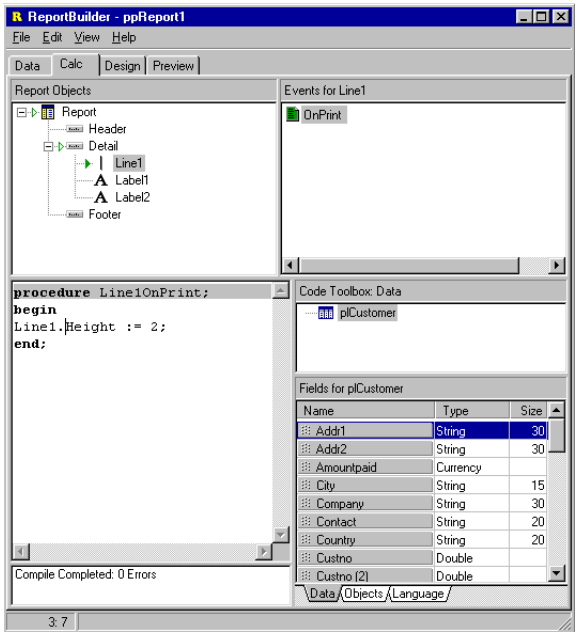
CODE

CODE

Introduction

Overview

RAP (Report Application Pascal) and the integrated development environment that can be used to create RAP programs are the result of the discovery that a visual solution for data processing and calculation was needed for end users. Even though RAP consists of an object-oriented compiler and extensible architecture 'under the covers', it appears as a simple, easy-to-use calc workspace within the Report Designer. The calc workspace is pictured below. The code editor window at the bottom of the workspace shows a simple event handler assigned to a line component.



The Report Application Programming language (or RAP for short) is designed to allow developers and end users to code calculations, event handlers, and stand-alone procedures for use with ReportBuilder Enterprise. RAP programs can be created, modified, compiled, and executed at run-time.

About RAP

RAP is easy to learn.
The RAP language is identical to Object Pascal. If you know how to code Delphi event handlers, you know how to code RAP; therefore, the learning curve for Delphi developers is minimal.

RAP lets you work with objects and object properties.
RAP provides full access to the report, bands, groups, report components, data pipelines, and any other objects you wish to pass along to the end user. New objects can be created. Both the published and public properties of objects are accessible, so end users can configure objects on-the-fly.

RAP is portable.
Once you've created a set of event-handlers or stand-alone procedures, you can save them as part of the report definition. You can export and import calculations from one report to another. You can even save calculations to a file or database BLOB field. And it can all be done at run-time, without recompiling your application.

RAP has a scalable user interface.

- For casual end users, a simple Calculations dialog is accessible from the speedmenu of the new TppVariable component. This dialog provides a place for calculations to be entered and returned via a single Result parameter. The calculation result is then displayed in the variable when the report prints.
- For more sophisticated users, RAP can be configured to display as an additional tab in the Report Designer. The new 'Calc' tab shows the bands of the report in a tree view, along with the associated variables. The end user can click on the variables and code the calculation in a syntax-sensitive edit window.
- The highest level of functionality is provided by the 'Event' view. This capability shows all of the report components in a tree view. When a component is selected, all of the events for the component are displayed. The user can then select an event and code an event handler.

RAP is extensible.
RAP is delivered standard with the ability to compile the most frequently used ReportBuilder and Delphi objects. If you want end users to access more than this, you can register additional RTTI information with RAP. RAP will then be able to compile your custom components. You can also extend the language by adding new, built-in functions and procedures which the users can call from their RAP programs. These procedures are written and compiled in Object Pascal and then registered with RAP.

RAP is optimized.
RAP is an add-on language for ReportBuilder Enterprise. However, ReportBuilder Enterprise does not require RAP in order to compile. You can install RAP and it will automatically appear in your Report Designer. The Delphi units that comprise RAP will only be added to your executable if you use RAP in your report.

RAP is cool.
The wide feature set, object-based functionality, and professional user interface makes RAP the language to beat for reporting applications.

The Calc Workspace

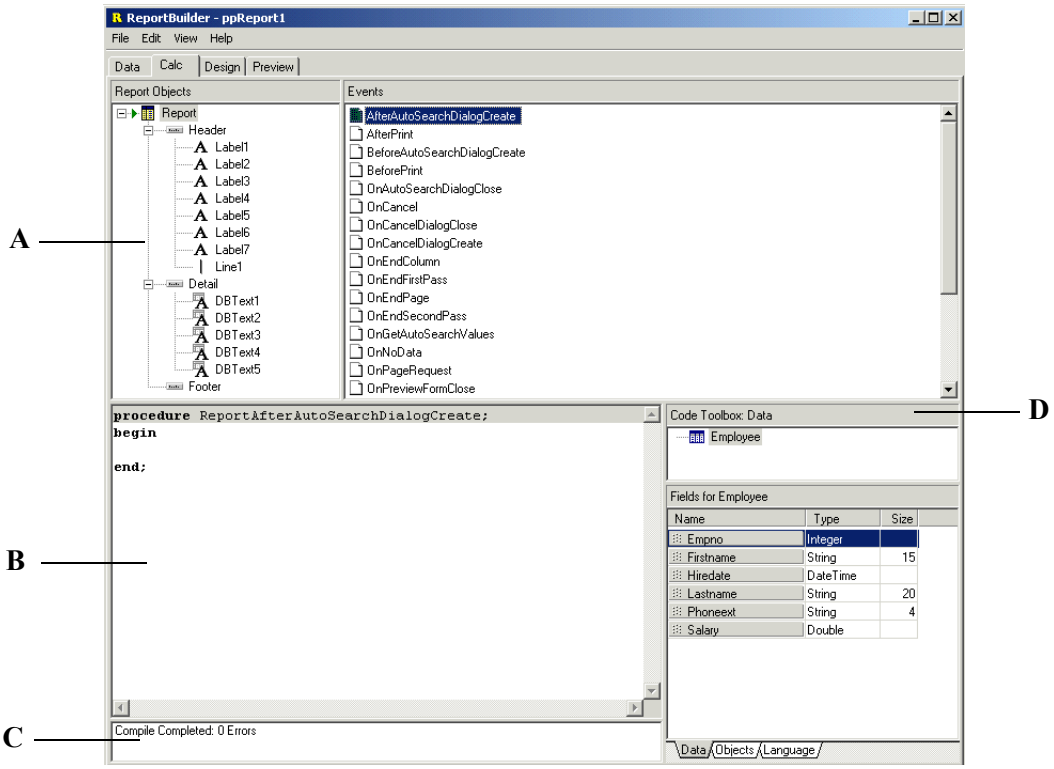
Overview

The Calc workspace is the development environment for RAP. With all of the RAP options available, the Calc workspace is a powerful Pascal development environment. For most end users, this may be more than they need. Judging your users' needs, you should define just how much of the RAP IDE to make available to them.

You can include the Calc workspace in your end user projects by adding raIDE to the uses clause of one of your units. When your users select the Calc tab, they will see the RAP IDE. The RAP IDE consists of:

- The Code Explorer
- The Code Editor
- The Message Window
- The Code Toolbox

The RAP IDE



A Code Explorer: Contains the tree view (left) and the list view (right)

B Code Editor: The space where you code and you compile the report

C Message Window: Messages from the compiler are presented here

D The Code Toolbox: Contains pieces of code that drag into the Code Editor

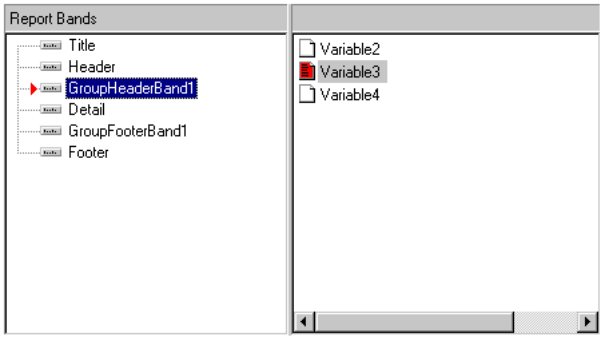
The Code Explorer

The Code Explorer is contained in the upper left and right panes of the Calc workspace.

The left pane contains a tree view — use this to navigate your report's code. The right pane contains a list view — it will display a variety of items depending on what is selected in the tree view. By right clicking on the tree you can display a context menu that allows you to control the behavior of the Code Explorer.

Variables View

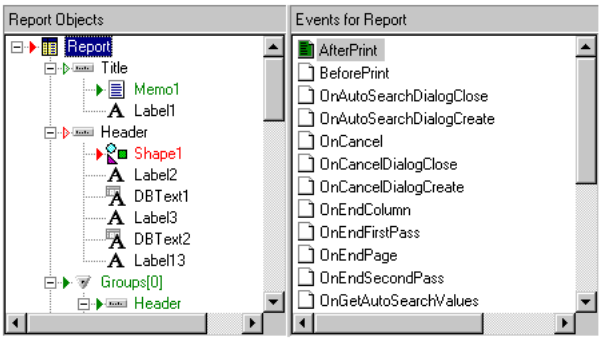
The Variables view of the Code Explorer is displayed by right-clicking the left pane and selecting Variables from the context menu.



The Variables view of the Code Explorer is displayed by right-clicking the left pane and selecting Variables from the context menu.

Events View

The Events view of the Code Explorer is displayed by right-clicking the left pane and selecting Events from the context menu.



This view displays a listing of all components contained within the report. The right pane displays any events associated with the currently selected component. Selecting an event will display the event handler, if one exists.

This view is good for viewing all report objects and their events.

In the treeview, you will notice small arrow shaped images to the left of some nodes. These are Compilation State Indicators. They are used to tell you where your code is and what state it is in. There are five possible indicators:

No symbol. Indicates that this component does not have any event handlers assigned, nor does it contain any components which have event handlers.

Red, not filled. Indicates that this component does not have any event handlers, but contains components which do. The red color indicates that one or more of the nested components has event handlers which do not compile.

Red, filled. Indicates that this component contains event handlers and that somewhere on this branch there is code that does not compile. If there is no code contained in components below this one, then the problem code is in this component. However, if there is code below this component, the problem may be in a child component's event handlers, in this component's event handlers or both.

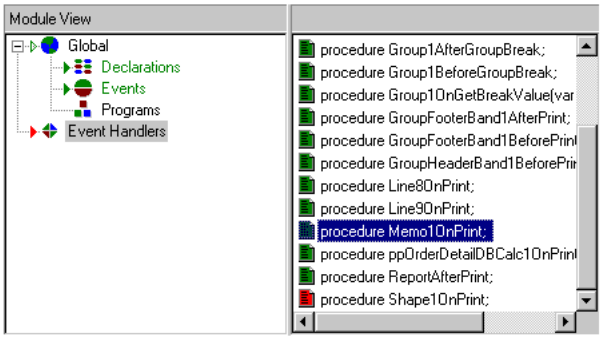
If a child component's code does not compile, the parent component will still display a red arrow even though its own code may compile.

Green, not filled. Indicates that this component does not have any event handlers assigned, but contains other components which do. The green color indicates that the event handlers of the contained components have compiled successfully.

Green, filled. Indicates that this component has event handlers assigned. The green color indicates that these event handlers, and any assigned to components nested within this one, have successfully compiled.

Module View

The Module view of the Code Explorer is displayed by right-clicking the left pane and selecting Module from the context menu.



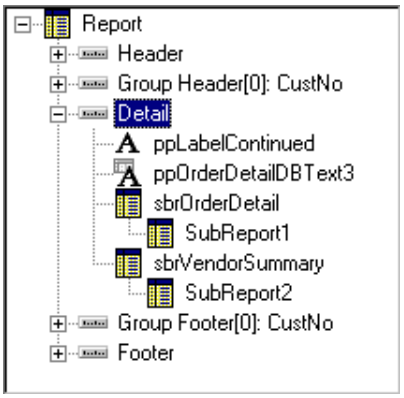
This view displays items which are visible to all event handlers of the report:

- **Declarations** – These are variables and constants that are globally visible throughout the report.
- **Events** – These are, in essence, the report's events. OnCreate and OnDestroy are good places for initialization and finalization code such as creating and freeing objects and initializing variables.
- **Programs** – These are procedures and functions that are globally visible throughout the report and can therefore be called from any event handler.
- **Event Handlers** – These are all event handlers that have been implemented in the report.

Global Scoping

Items declared in the Global section are effectively visible throughout the module and to all child subreports. When compiling, the compiler tries to resolve all references within the current module. If it can't find an identifier, it begins looking in the next parent report's global section and continues this up to the main report.

Consider the following scenario involving multiple level subreports:



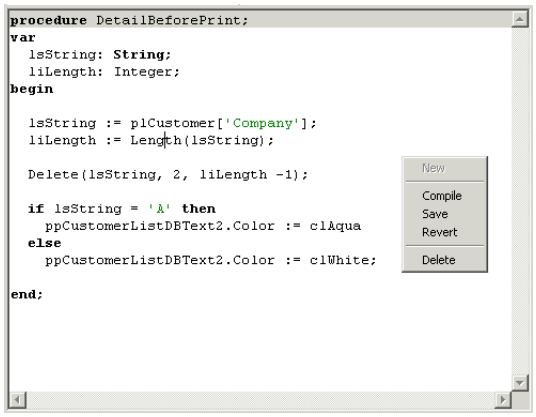
Suppose we declare, in the main report's Global section, a variable, MyGlobalString: String; This variable would be visible throughout the main report and all subreports as the compiler will eventually find the declaration in the parent of any subreport.

However suppose, again, that we declare MyGlobalString in the Global section of sbrOrderDetail. Now the variable is visible to sbrOrderDetail and to SubReport1 but not visible to sbrVendorSummary, SubReport2 or even the main report.

Just keep in mind: declarations are visible to any child subreports of the current report.

The Code Editor

The Code Editor is the place where you actually write and modify RAP code.



This syntax-sensitive editor is similar to the one found in Delphi, with the exception that it displays only one procedure at a time. Whatever the currently selected item is (event handler, procedure, function, variable declarations, etc.), only the code for that item is displayed in the Code Editor. When an item is selected, the editor will either contain the code implementation, or will be blank if no implementation exists. If no implementation exists, you can create one by clicking in the editor. You can then enter code by either typing or by dragging items from the Code Toolbox and dropping them into the editor. If an item is dropped into the editor over a section of selected code, the selected code will be replaced.

The Code Editor's context menu contains the following items:

New

New has the same effect as clicking in the Code Editor. It is only enabled if there is no implementation for the item currently selected in the Code Explorer.

Compile

Compile activates the RAP compiler to attempt to compile the current procedure and any procedures upon which the current one depends.

Save

The Calc workspace maintains an intermediate buffer for the Code Editor. Selecting Save will commit the current contents of the Code Editor to the buffer; it will not save the entire report. Selecting Save has the same effect as navigating away from, and then returning to the current procedure.

Revert

Use Revert to replace the contents of the Code Editor with what is currently contained in the code buffer. This has the effect of removing all changes since the last save.

Delete

Select Delete to remove the current procedure entirely.

Message Window

The Message Window functions in essentially the same manner as Delphi's message window. Messages from the compiler are presented here. You can navigate to the location of compiler errors by double-clicking the error message.

The Code Toolbox

The Code Toolbox is a visual code repository. It contains most of the identifiers and code elements that the RAP compiler recognizes.

The Code Toolbox enables you to:

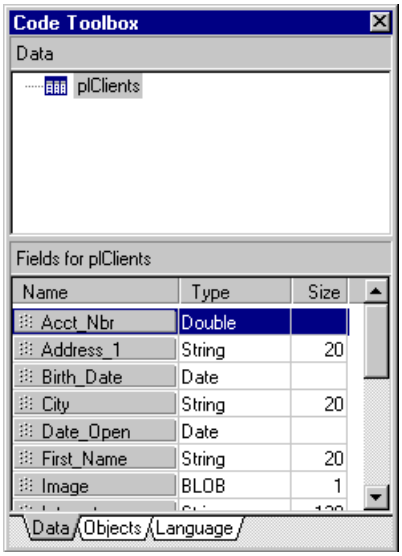
- View identifiers grouped by available data, visible object properties or language features
- Generate code by dragging identifiers into the Code Editor. Functions dragged into the editor will generate a place-holder parameter list. For example, the following code is generated when you drag the Copy function into the Code Editor:

```
Copy(S, Index, Count);
```

Each tab of the Code Toolbox consists of a treeview and a list of identifiers. The treeview allows you to navigate groups of identifiers listed on each tab.

Data Tab

The Data tab of the Code Toolbox displays data pipelines and fields, allowing you to drag and drop field references into the Code Editor.



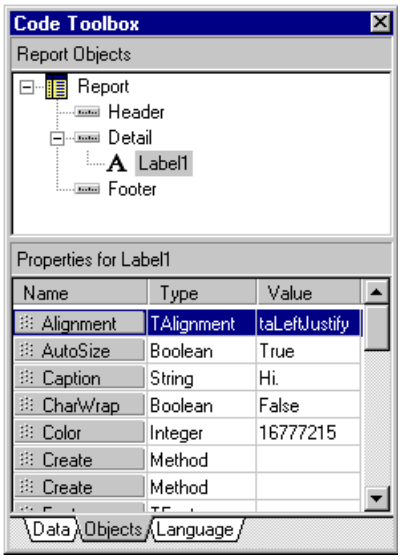
Selecting a pipeline from the list will display all of the fields in that pipeline as well as data type and size information for the fields.

To insert a field value into the code editing window, select the field and drag it into the Code Editor. The code necessary to retrieve the field value from the pipeline will be generated. For example, dragging the 'City' field from the Code Toolbox pictured above would result in this code:

```
Value := Clients['City']
```

Objects Tab

The Objects tab of the Code Toolbox displays report objects and their properties, allowing you to drag and drop properties into the Code Editor.



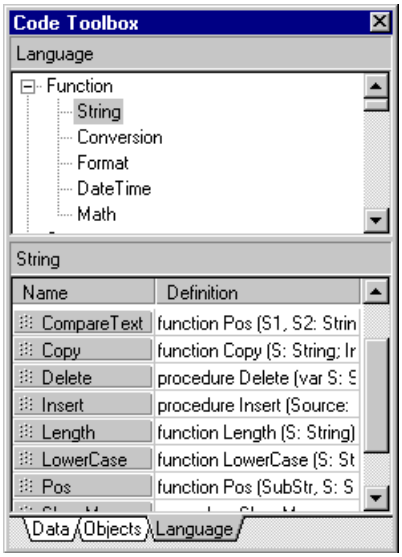
Selecting an object from the tree will display a list of that object's properties.

To insert a property into the Code Editor, select the property and drag it into the Code Editor. The code necessary to retrieve the value of the property or call the method will be generated. For example, dragging the 'AutoSize' property from the Code Toolbox pictured above would result in the following code:

```
Label1.AutoSize
```

Language Tab

The Language tab of the Code Toolbox displays RAP language elements, allowing you to drag and drop elements into the Code Editor.



Selecting a category from the tree will display a list of elements for that category.

To insert an element into the Code Editor, select the element and drag it to the Code Editor. The code necessary to reference or use the element will be generated. Note that when you drop an item such as a function call, the function's parameter list is provided. For instance, if you drag Copy into the Code Editor, it will expand as:

```
Copy(S, Index, Count);
```

When you register pass-through functions with RAP, they will appear automatically in the Function section of the Language Tab. Identifiers listed in the Toolbox will also display relevant information such as Type, Size, Signature, etc. Identifiers which appear in the Code Toolbox are registered with RAP via RAP's extensible architecture. You can use this architecture to register your own pass-through functions and objects with RAP — this allows your new functions to appear in the Code Toolbox and to be recognized by the compiler.

Writing RAP Code

Programs in RAP

All code in a report is contained in the Code Module. The Code Module contains all the event handlers for objects contained in the Report. There is also a global section that can contain module level events, declarations, procedures and functions. This part of the Code Module is visible from the Module view of the Code Explorer.

The programs and variables declared in the global section of a module are visible to all subreport code modules in the report. Event handlers appear much as they would in Delphi, with the exception that the Sender parameter is omitted from the signature — this is due to the fact that event handlers can only be assigned to one object in RAP, whereas Delphi allows the same event handler to be assigned to multiple events.

Coding an Event Handler

In order to code a new event handler, simple select an event in the Code Explorer and click in the white space of the Code Editor. Both the signature and the begin/end pair for the event handler will be generated automatically. You can then begin entering your code. A typical event handler would appear as:

```
procedure Variable1OnCalc (var Value: Variant);
begin
end;
```

Note: When you are working in the Variables view of the Code Explorer, the event handler signature and begin/end pair will not appear in the Code Editor. You will see only a single line for assigning the Value:

```
Value :=
```

In this mode the Code Explorer displays only variables and is, in a sense, equating variables with the value returned by the OnCalc event. This is valuable for limiting the amount of functionality you wish to reveal to your users.

Compiling Event Handlers

The RAP compiler attempts compilation of the entire module automatically when any of the following happen:

- You load a report.
- You select the Calc tab within the Report Designer.
- You switch views in the Code Explorer.

It is also possible to compile the currently selected event handler. To do this, right-click over the Code Editor and select Compile from the popup menu. The compiler will check the global sections and any other programs needed to compile the current event handler.

Declaring Local Variables

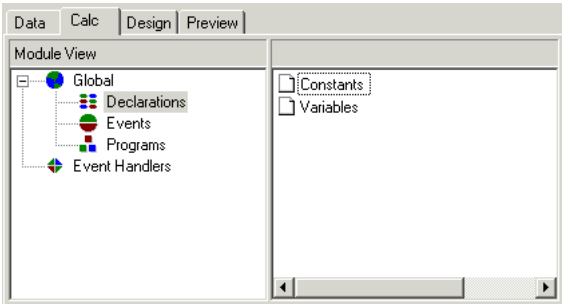
Local variable declarations in RAP are just the same as in Object Pascal. To add a local variable declaration to a function, activate the Code Editor for the current item and place the cursor between the function's declaration and the begin. Type var and declare your variables just as you would in Delphi.

Declaring Local Constants

Local constant declarations in RAP are just the same as in Object Pascal. To add a local constant declaration to a function, activate the Code Editor for the current item and place the cursor between the function's declaration and the begin. Type const and declare your constants just as you would in Delphi.

Declaring Global Variables

To declare a global variable, right-click on the Code Explorer and select Module. The treeview will change to display the global section. Click on the Declarations node – this will display two items in the listview: Constants and Variables.



Select the Variables item. If you have already declared some variables, they will be displayed in the Code Editor. If the Code Editor is blank, right-click the Variables item and select New – this will activate the Code Editor and add **var** to the first line.

Declare your variables using standard Object Pascal syntax.

```
var
  myGlobalString: String;
  myGlobalInteger: Integer;
  myStringList: TStringList;
```

Declaring Global Constants

To declare a global constant, right-click on the Code Explorer and select Module. The treeview will change to display the global section. Click on the Declarations node – this will display two items in the listview: Constants and Variables.

Select the Constants item. If you have already declared some constants, they will be displayed in the Code Editor. If the Code Editor is blank, right-click the Constants item and select New – this will activate the Code Editor and add const to the first line.

Declare your constants using standard Object Pascal syntax.

```
const
  csMyConstString = 'What is hip?';
  ciMyBigAnswer = 42;
```

Declaring Global Procedures and Functions

The global section of the Code Module can contain functions visible throughout the module and to any subreports below the current report. To declare such functions, right-click on the Code Explorer and select Module. The treeview will change to display the global section. Click on the Programs node – the listview will display any extant functions. If you have not declared any, the listview will be empty.

Right-click on an empty space in the listview – note that the first two menu items are New Function and New Procedure. Selecting either of these items will create a declaration and an implementation stub.

If you select New Function, a new function named GlobalFunction1 will be added to the listview and the following implementation will be added to the Code Editor:

```
function GlobalFunction1: Variant;
begin

  Result :=

end;
```

Likewise, if you select New Procedure, a new procedure named GlobalProcedure1 will be added to the listview and the following implementation will be added to the Code Editor:

```
procedure GlobalProcedure1;
begin

end;
```

Procedure and Function Parameters

Function parameter lists are the same in RAP as in Object Pascal.

Because event handlers in RAP cannot apply to more than one event, event handlers in RAP do not have a Sender parameter in their signature.

Calling Procedures and Functions

Calling a function in RAP is no different than in Object Pascal. As long as the function is in scope, you can call it.

When calling a parent report's global function from within a subreport, you do not need to qualify the identifier with the parent report's name.

In other words, if your Main report has a global procedure, GlobalDoSomething, and you want to call it from within SubReportA's OnCreate event, you do not have to say Main.GlobalDoSomething. Merely calling GlobalDoSomething will suffice.

Configuring RAP

End User Configurations

When building an end user solution, you must decide how much of RAP's functionality to expose to your users. RAP allows a continuum from a calculations dialog to a full development environment. Likewise, you can give your users only the base language features included in RAP, or you can expose new, custom pass-through functions, make RAP aware of your own classes and even allow users to display custom forms you design.

The two main properties used for scaling the RAP environment are TppDesigner.RAPOptions and TppDesigner.RAPInterface.

The RAPInterface property controls how the Calc workspace will be presented to the user — as a dialog, a tab or both.

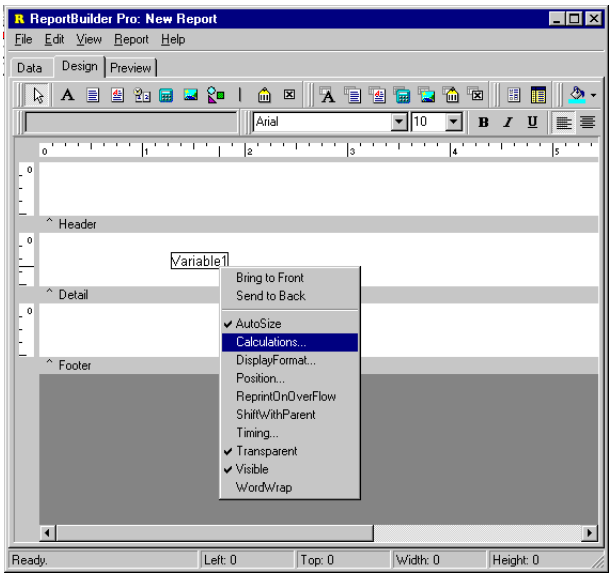
When RAP is presented as a Calc tab in the Report Designer, RAPOptions allows you to specify how much the user is able to view and edit.

The RAP user interface can be configured to meet the needs of your end users. There are two basic configurations available:

1 Calculations Dialog

For casual end users, a simple Calculations dialog is accessible from the speedmenu of the TppVariable component. This dialog provides a place for calculations to be entered and returned via a single Result parameter. The calculation result is then displayed in the variable when the report prints.

In order to configure the user interface to provide this level of functionality, set the RAPInterface riNotebookTab property to False and the riDialog property to True. The figure below illustrates the Calculations... option available when the end user accesses the speed menu of a variable.

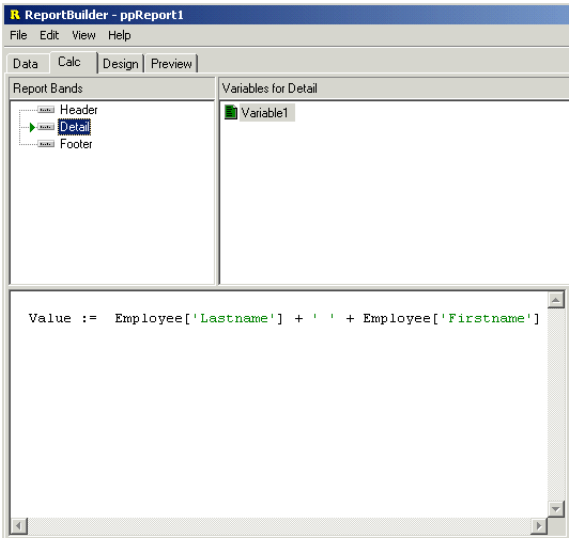


2 Calc Tab

In this configuration, your users have the same functionality as in the Calc Dialog, but it is available in the Calc tab. In the Calc dialog, the Code Explorer is not shown since, in that dialog, we deal with only one variable at a time. However, in the Calc tab, the explorer is shown. In order to configure the user interface to provide this level of functionality, set the RAPInterface riNotebookTab property to True and set all of the RAPOptions to True. When the Report Designer is displayed, the Calc tab will be visible. If you right-click on the treeview on the left, a speed menu that shows the different views of calculations available will be displayed:

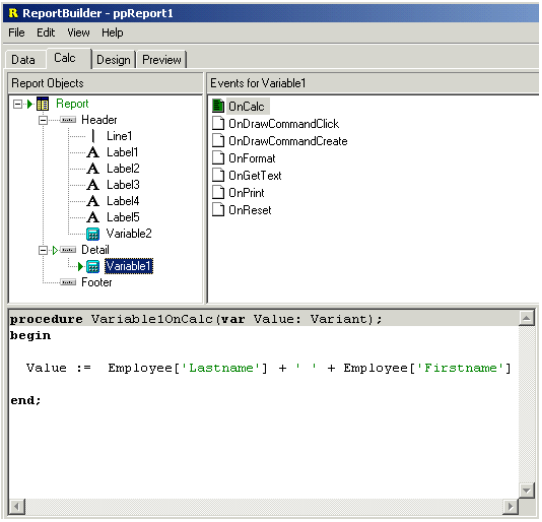
A Variables

This view shows the variables in each band of the report and any variables contained in the currently selected band. The Calc tab makes it easier to see the code associated with any variables in the report, but still keeps the concept of events hidden from the user. Selecting a variable displays the OnCalc event handler, if one exists.



B Events

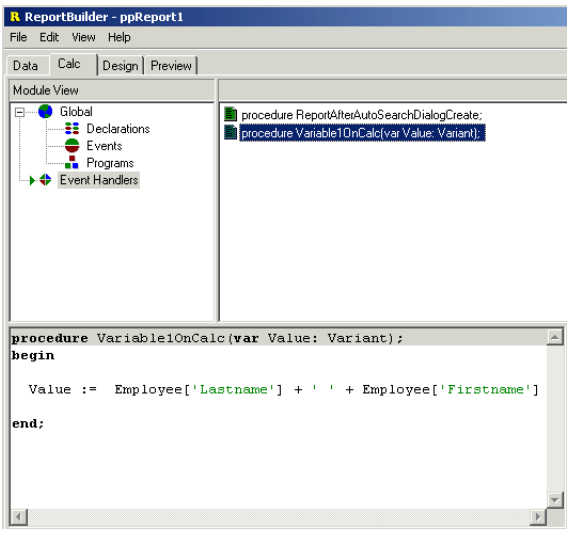
This view shows the report and all of the objects within it. The user has access to all the report objects' events, and can code event handlers for them. The right pane displays any events associated with the currently selected component. Selecting an event will display the event handler, in one exists. An event handler can be coded for any object. This view is good for viewing all report objects and their events.



C Module

Finally, to expose all the power of the Calc workspace you can add the global settings to RAPOptions. This allows the user to see and edit everything available in RAP. This view displays items which are visible to all event handlers of the report: global declarations, functions, programs, and event handlers.

- **Declarations** - The variables and constants that are globally visible throughout the report.
- **Events** - These are, in sequence, the report's events. In the case where the preview window is displayed, OnCreate and OnDestroy fire when the window is opened and closed, respectively. This is different from Report.BeforePrint and AfterPrint in that those methods will fire each time Report.Print is called. OnCreate and OnDestroy are good places for initialization and finalization code such as creating and freeing objects and initializing variables.
- **Programs** - The procedures and functions that are globally visible throughout the report and can therefore be called from any event handler.
- **Event handlers** - All event handlers that have been implemented in the report.



Extending RAP

RAP Pass-Through Functions

At the heart of RAP is a run-time compiler that executes your users' code independent of Delphi's compiler. Therefore RAP's compiler has its own independent set of recognizable tokens. In other words, RAP is a subset of Object Pascal.

The method for making the RAP compiler aware of functions is to declare descendants of TraSystemFunction – one for each function – and to register them with the compiler using raRegisterFunction. Each descendant class overrides ExecuteFunction – this method tells the RAP compiler what to do when a function name is encountered in the users' code, thus passing-through the true Delphi functionality to the end user.

For example, when the RAP compiler comes upon the Copy function, it executes the ExecuteFunction method which, in essence, tells RAP how to assign the function's parameter list, then calls Delphi's Copy function.

For examples of registering new pass-through functions, see the main RAP demo: RBuilder\Demos\0. RAP\1. Main\RAP.dpr – reports #31-33. Also see the Adding Functions Tutorial.

Adding Functions to the Code Toolbox

While the Language tab of the Code Toolbox contains many useful functions for you and your user, it is sometimes necessary to add functionality. You can register a new pass-through function with RAP quite easily. The steps are:

1 Identify a Delphi function to call. This can be a function you have written, a DLL call, a Win API call or a standard VCL function.

2 Create a TraSystemFunction descendant.

3 Return the signature of your procedure in the GetSignature method.

4 Indicate whether the call HasParams and IsFunction via these two boolean methods.

5 Make the actual call to the Delphi function or procedure in the ExecuteFunction method.

6 Register your new function with RAP via a call to raRegisterFunction. Place this call in the initialization section of the unit containing the descendant class.

7 Add this unit to your project.

When you run the application, you will see your new function in the language tab of the Code Toolbox. Also, you will be able to call the function from any RAP event handlers or programs you create. For a detailed tutorial, see Adding New Functions to RAP.

Adding Classes and Properties via RTTI

The report components in the RCL are all surfaced in RAP. In addition, you will notice that the compiler knows about TStrings, TStringList and TList. It is possible to make RAP aware of other objects you may wish to expose within the Calc workspace. For instance, you might want your users to be able to refer to TmyUniversalFinancialComponent.

To do so, see Extending the RAP RTTI. This tutorial will show you not only how to expose your new class, but how to make that class's public properties and methods available as well.

Debugging RAP Code

CodeSite Support

There is no integrated debugging for RAP – that's a bit beyond the scope of the product. However, there is an option to provide support for CodeSite – Raize Software Solutions' excellent debugging tool.

In order for you, as a developer, to use this option, you must have a licensed copy of CodeSite installed on your computer. If you wish to provide this support for your users, your users must have licensed copies of CodeSite installed.

RAP's support of CodeSite in no way changes the licensing agreement of Raize Software Solutions; anyone utilizing RAP's support of CodeSite must have a properly registered version of CodeSite.

Now, that being said, in order to enable the CodeSite support at run-time, you must add the raCSFuncs unit to your uses clause. This will register the CodeSite pass-through functions with RAP, thus making them available in the Code Toolbox.

To enable the CodeSite support at design-time, you will need to compile and install a package into Delphi. See the ReadMe.doc file in your RBuilder\Demos\0. RAP\2. CodeSite directory for instructions. Most of the CodeSite calls are made available, some with a few modifications due to RAP's architecture.

A few notable items :

- CodeSite's Enabled property is available as the csEnable procedure in RAP. Pass in a boolean value to enable or disable CodeSite.
- Since RAP does not yet support Record types, the SendPoint and SendPointEx functions were changed to accept integers, X and Y instead of a TPoint and the SendRect and SendRectEx functions were changed to accept integers, Left, Top, Right and Bottom instead of a TRect. The pass-through functions will, in turn, map these integers into the proper values when calling the corresponding CodeSite methods.
- There is a demo project showing the use of the CodeSite pass-through functions in RBuilder\Demos\0. RAP\1. CodeSite.

Using the CodeSite Functions

For those unfamiliar with CodeSite, here are some ideas for how to use it with RAP. Note that this is not a primer on using CodeSite; see the CodeSite documentation for that information.

Let us say, for instance, that a TppVariable is not displaying a value you think it should and you want to make sure that a section of code is actually being executed. You might add the following code to the variable's OnCalc event:

This is a fairly simple example, but you can see that the liberal use of CodeSite calls can be essentially like stepping through code, looking at values.

There is a demo project showing the use of the CodeSite pass-through functions in RBuilder\Demos\0. RAP\2. CodeSite.

Code	Variable1OnCalc Event

```
procedure Variable1OnCalc(var Value: Variant);
var
  lsLabelText: String;
begin

  csEnterMethod('Variable1OnCalc');
  csSendString('TaxRate', plCustomer['TaxRate']);
  if plCustomer['TaxRate'] > 8 then
    begin
      csSendBoolean('Detail.Overflow', Detail.Overflow);
      if Detail.Overflow then
        begin
          lsLabelText := 'Continued...';

          Value := '';
        end
   else
        begin
          lsLabelText := plCustomer['Company'];
          Value := GetMyValue;
        end;
      csSendString('Value', Value);
    end
  else
    begin
      csSendWarning('Invalid TaxRate');
      Value := 'Invalid';
    end;
  csExitMethod('Variable1OnCalc');

end;
```

Conditionally Compiling CodeSite Support

Repeatedly adding and removing CodeSite calls can be troublesome, especially if you have many of them riddling your code. For that reason, the CodeSite calls can be conditionally compiled. All of the CodeSite functions in the raCSFuncs unit have conditional statements around their ExecuteFunction implementations and at the beginning of the unit is the line:

```
{x$DEFINE CODESITE}
```

In order to enable the function implementations, remove the "x" from the beginning of the line. To disable them, add the "x". This will allow you to leave all of the CodeSite calls in the code without activating the CodeSite object on your user's machine.

In other words, to enable CodeSite support for your report, do the following:

1 Open the raCSFuncs unit.

2 Scroll to the top of the unit, just below the interface clause.

3 Remove the 'x' from the line:

```
{x$DEFINE CODESITE}
```

This will enable CodeSite support.

4 Save the unit.

5 Rebuild your project by selecting Build All in Delphi.

To disable CodeSite support for your report while not removing the CodeSite calls themselves, do the following:

1 Open the raCSFuncs unit.

2 Scroll to the top of the unit, just below the interface clause.

3 To enable CodeSite support, add an ’x’ at the beginning of the line between the '{' and the '$'.

```
{$DEFINE CODESITE}
```

4 Add an 'x'

5 Save the unit.

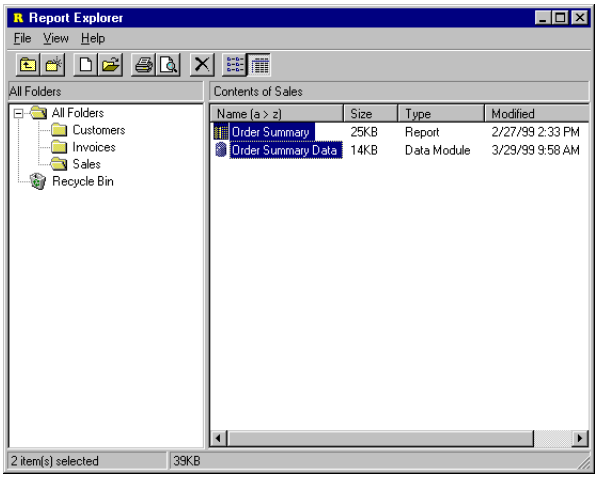
6 Rebuild your project by selecting Build All in Delphi.

DESIGN

DESIGN

The Report Explorer

The Report Explorer component allows you to deploy a Windows Explorer interface that your end users can use to organize their reports. The Report Explorer handles all operations via data pipelines; therefore, the folder structure and all of items within it can be saved to database tables. This interface presents a minimal learning curve, as most users are familiar with the operation of the Explorer. The Report Explorer user-interface is pictured below:



Report Explorer Toolbar

The action that each button on the Report Explorer toolbar performs is described below.

Up One Level

When a folder is selected, use this button to select the parent folder.

New Folder

Creates a new folder within the currently selected folder.

New Report

Creates a new report and displays the Report Designer.

Open Report

Opens the currently selected report and displays it in the designer.

Print

Prints the currently selected report.

Print Preview

Opens the currently selected report and displays it in the print preview window.

.

PRINT

PRINT

End-User Options

Overview

You can control whether end users can Archive or Print to any of the supported file formats using the following properties of the Report component:

- AllowPrintToArchive
- AllowPrintToFile

Setting AllowPrintToArchive to True will enable end users to archive reports. You would then need to provide a user interface for previewing and printing the archived files using the ArchiveReader component. For an example of how to use the ArchiveReader, see example 151 (..\RBuilder\Demos\Reports\dm0151.pas).

Setting AllowPrintToFile to True will enable end users to print to the any of the available file formats. You can control which file formats are available to end users via the device class registration procedures. For example, the following code would remove the ReportTextFileDevice from the list of available file formats:

```
uses
   ppDevice, ppFilDev;

   ppUnRegisterDevice(TppReportTextFileDevice);
```

End-User Applications

In order to set the AllowPrintToArchive and AllowPrintToFile properties within the context of an end-user application, the OnNew and OnLoad events of the Report.Template object should be used to set the default properties of the report. For example, you would add the code below to the main form in order to create the end-user application.

Code | *Coding the main form of an end-user application*

```
procedure TmyEndUserSolution.FormCreate(Sender: TObject);
begin

   {assign event-handler to template events}
   ppReport1.Template.OnNew     := ReportTemplateEvent;
   ppReport1.Template.OnLoadEnd := ReportTemplateEvent;

   {remove the report text file device }
   ppUnRegisterDevice(TppReportTextFileDevice);

end;

procedure TmyEndUserSolution.ReportTemplateEvent(Sender: TObject);
begin

   {set default report properties}
   ppReport1.AllowPrintToArchive := True;
   ppReport1.AllowPrintToFile    := True;

end;
```

DEPLOY

DEPLOY

Summary

Overview

There are two totally different but vital issues involved in deciding how you will deploy your application:

1 How to deploy ReportBuilder as part of your application

2 How to deploy the reports created with ReportBuilder as part of your application

Application Deployment

When creating your application exe, you can choose to either compile everything into one executable, or you can choose to distribute the packages provided with Delphi, ReportBuilder, or any other components you are utilizing in your application as separate files. The latter option is referred to as 'distributing with packages' (packages here refers to a Delphi compatible DLL) and results in a smaller, more efficient executable file.

These two approaches for deploying your application are discussed later in this section.

Report Deployment

The second part of deployment involves determining how to deploy the reports created with ReportBuilder as part of your application. ReportBuilder provides the most flexible deployment options of any reporting product on the market. The most commonly used deployment strategies are as follows:

1 End-User Reporting Solutions

The end-user reporting solution uses the Report Explorer to manage reports stored in a database. There are two deployment options for distributing the end-user database tables.

A Local End-User Database

In this scenario, each end user has a local copy of the end-user tables that are used as a personal database for storing reports.

B Shared End-User Database

In this scenario, there is a single copy of the end-user tables that are used as a common database for storing all reports.

2 Standard Reporting Solutions

A standard reporting solution refers to one in which the developer creates reports that may be executed by the end user.

A Compile reports into the executable.

Allow the report component to reside in an invisible form along with the necessary data access objects. When it is time to print the report, instantiate the form and call the Print method. You could slightly vary this architecture by using a data module as a container for the data access objects. This method requires the form to contain the ppReport component.

B Compile reports into a package.

Similar to option A, but rather than compile the forms containing the reports into the executable, the report forms are compiled into a package. The advantage of this approach is that it optimizes the size of the executable and dynamically loads the package containing the reports as needed.

C Distribute report template files.

Place all the data access components for the reports in a data module. Place a single report component that uses the data module on the form. Create all of the report layouts via this one report component, saving each one down to a separate report layout file. When it is time to print the report, load the report from file and call the Print method.

D Distribute a report database.

Use all of the same steps as option C, but save the report definitions to a database BLOB field instead of separate files. With this approach you can save all of your report definitions in one database table.

A full-featured implementation of Option A above is provided in the 'Building a Reporting Application' tutorial in the Developer's Guide. Option A has the advantage of requiring the least code. The main advantage of options C and D is that you can change report layouts and redeploy them to your end users with regenerating your executable. Because Option A compiles the report layouts into the executable, it requires a recompile any time a report must be changed. In terms of report layout load time or print speed, all options are essentially equal.

REPORT TUTORIALS

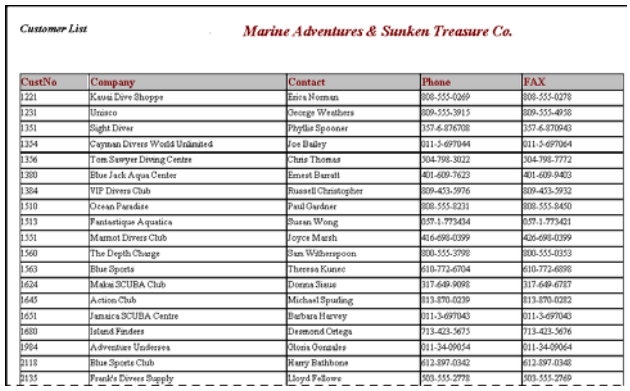
REPORT TUTORIALS

Creating a Report Via the Data Tree

Overview

This tutorial will show you how to do the following:

- Use the Data Tree
- Work with the Report Designer
- Create and configure the most common report components



Customer List *Marine Adventures & Sunken Treasure Co.*

CustNo	Company	Contact	Phone	FAX
1221	Kauai Dive Shoppe	Eric Norman	808-555-0269	808-555-0278
1231	Unisco	George Weathers	809-555-3915	809-555-4938
1351	Sight Diver	Phyllis Spooner	357-6-876708	357-6-870943
1354	Cayman Divers World Unlimited	Joe Bailey	011-3-697044	011-3-697064
1356	Tom Sawyer Diving Centre	Chris Thomas	304-798-3022	304-798-7772
1380	Blue Jack Aqua Center	Ernest Burrett	401-609-7623	401-609-9403
1384	VIP Divers Club	Russell Christopher	809-453-5976	809-453-5932
1510	Ocean Paradise	Paul Gardner	808-555-8231	808-555-8450
1513	Fantastique Aquatica	Susan Wong	65-1-775434	65-1-775421
1551	Marmot Divers Club	Joyce Marsh	416-698-0399	416-698-0399
1560	The Depth Charge	Sam Witherspoon	800-555-3998	800-555-0353
1563	Blue Sports	Theresa Kunec	610-772-6704	610-772-6898
1624	Make SCUBA Club	Donna Sims	317-649-9098	317-649-6787
1645	Action Club	Michael Spurling	813-870-0239	813-870-0282
1651	Jamaica SCUBA Centre	Barbara Harvey	011-3-697043	011-3-697043
1680	Island Finders	Desmond Ortega	713-423-5675	713-423-5676
1984	Adventure Undersea	Gloria Gonzalez	011-34-09054	011-34-09064
2118	Blue Sports Club	Harry Bathbone	612-897-0342	612-897-0348
2135	Frank's Divers Supply	Lloyd Fellows	303-555-2978	303-555-2969

Create a Tutorial Folder

1 Access the Windows Explorer.

2 Create a folder on the root of your hard drive.

3 Name the folder My RB Tutorials.

Note: When installing new versions, the RB installation program will delete the existing RB install directory. For this reason, we suggest creating the tutorials directory either on the root or somewhere outside of the directory for RB.

Create a New Application

1 Run Delphi.

2 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

3 Set the Form name to 'frmViaDataTree'.

4 Select File | Save As from the Delphi menu and save the form under the name rbViaDT in the My RB Tutorials directory.

5 Select View | Project Manager from the main menu.

6 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

7 Save the project under the name rbDTProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create a Table, DataSource, and DataPipeline Component

1 Select the Data Access tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblCustomer
TableName	customer.db

4 Add a DataSource component to the form.

5 Configure the DataSource component:

DataSet	tblCustomer
Name	dsCustomer

6 Select the RBuilder tab of the Delphi component palette.

7 Add a DBPipeline component  to the form.

8 Configure the DBPipeline component:

DataSource	dsCustomer
Name	plCustomer

Create a Report and Connect it to the Data

1 Add a Report component  to the form.

2 Configure the Report component:

DataPipeline	plCustomer
Name	rbCustomerList

Invoke the Report Designer and Set the Paper Orientation

1 Double-click on the Report component to display the Report Designer.

2 Size and move the Report Designer window so that the Object Inspector is visible.

3 Select the File | Page Setup option from the Report Designer main menu.

4 Click the Paper Size tab and change the paper orientation from Portrait to Landscape, then click the OK button.

Set the Header Band Height to 1 inch

1 Place the mouse over the gray divider entitled '^ Header'. The cursor should change to a double-sided arrow. Hold down the left mouse button, drag down a short distance, and release the mouse button. Check the vertical ruler on the left and drag down until you can see the 2 inch mark.

2 Drag the band divider up until the top guide indicates you have reached the 1 inch mark.

Note: As you drag the band divider downward to increase the height, the height of the ruler grows with it. As you drag the band divider upward, a guide is displayed on the ruler to indicate the new height of the band.

3 Right-click over the white space of the header band and access the Position dialog. The value for the height should be 1.

Create Labels in the Header Band

1 Click the Label component icon on the Report Designer component palette.

2 Click on the left side of the header band. A label will be created in the header band. It will become the current selection in the Object Inspector.

3 Locate the Edit box at the upper left corner of the Report Designer. Select the text inside it and type Customer List.

4 Locate the font controls on the Format bar.

Note: The font controls appear to the right of the Edit box and provide the following functions:

- Font Name drop-down list
- Font Size drop-down list
- Bold
- Italic
- Underline Font Style
- Text Alignment
- Font Color
- Highlight Color

5 Set the font:

Font Name	Times New Roman
Font Size	12
Font Style	Bold & Italic

6 Position the label at the upper left corner of the header band.

7 Place another label near the center of the header band.

8 Configure the label:

Caption	Marine Adventures & Sunken Treasure Co.
Font Name	Times New Roman
Font Size	16
Font Style	Bold & Italic
Text Alignment	Centered

9 Set the font color to maroon by clicking on the right side of the Font Color icon to display the drop-down Color Palette, then clicking on the maroon colored square.

Note: When you select a color via the Font or Highlight Color icons, the selected color is displayed below the icon. You can then click on the icon directly (without re-displaying the drop-down color palette) and the currently selected component or components will be set to that color.

10 Position the Marine Adventures label at the top of the header band workspace area.

11 Select View | Toolbars | Align or Space from the Report Designer main menu. The toolbar should appear as a floating window.

12 Click and drag the toolbar window to the left side of the Report Designer and position it so that it docks to the left of the vertical ruler.

13 Center the label by clicking the Center Horizontally icon on the Align or Space toolbar.

layout check

Use the Data Tree to Lay Out the Customer Information

1 Select View | Toolbars | Data Tree. Position the Data Tree to the left of the Report Designer.

2 Click the Layout tab at the bottom of the Data Tree and configure the drag-and-drop settings as follows:

Create	All
Style	Tabular
Label Grid	True
Label Font Name	Times New Roman
Label Font Style	Bold
Label Font Size	12 point
Label Font Color	Maroon
Field Grid	True
Field Font Name	Times New Roman
Field Font Style	Regular
Field Font Size	10 point
Field Font Color	Black

Note: In this tutorial we are using the Tabular style to create a grid of labels and field components. After you have completed this tutorial, you should try it again using the Vertical style as well as various other combinations of the available layout settings.

3 Click on the Data tab of the Data Tree and select the Customer DataPipeline from the tree list at the top. The Customer fields will be displayed in the list at the bottom.

4 Click on the CustNo field and then hold down the Ctrl key and click on the following fields in the order specified:

Company
Contact
Phone
FAX

Note: This method of field selection is called the Ctrl-click method.

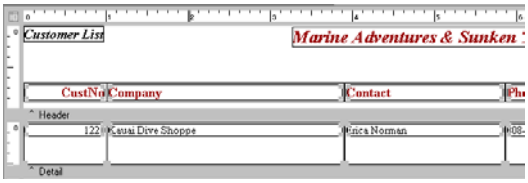
5 Drag the selected fields onto the workspace, positioning the mouse at the bottom left corner of the header band. Release the mouse button. A grid of labels will be created at the bottom of the header band component and a corresponding grid of DBText components will be created in the detail band.

Note: The Data Tree has a built-in behavior for creating labels and data-aware components. If the components cannot fit in the band's vertical space below the label, they are stacked at the end of the band.

Note: The width assigned to each DBtext component is calculated using the DisplayWidth of the data pipeline's field components. When using the DBPipeline, the field components are created automatically and use the DisplayWidth information of the fields in the dataset to which the pipeline is connected. You can override this behavior by setting the AutoCreateFields property to False and using the DataPipeline's Fields Editor to set the default DisplayWidth.

6 Close the Data Tree window.

layout check



Adjust the Layout

1 Scroll right until you can see the right edge of the header band.

2 Click in the white space of the header band to deselect the components.

3 Scroll back to the left, select the CustNo Label, and then hold down the Shift key and click the corresponding DBText component directly below it. This method of component selection is called the Shift-click method.

4 Click on the Left Justify icon.

Note: The CustNo field is numeric; therefore, the label and DBtext component were right justified when created by the Data Tree.

5 Click on the white space of the header band, just above the CustNo label. Hold down the mouse button and move the mouse cursor down to about the middle of the CustNo label and then move the mouse to the right - across all of the labels and shapes in the header band.

6 Release the mouse button to select the objects.

Note: A dotted box will be drawn on the workspace as you move the mouse. When you release the mouse button, all objects that cross the boundary of the box will be selected. This method of component selection is called the bounding box method.

Note: If you cannot select all objects because you need to scroll to the right, then select as many as you can and release the mouse button. Now scroll to the right. Hold down the Shift key and create another bounding box by pressing the left mouse button and dragging the mouse again. When you release the mouse button, the selected objects will be added to the list. If you reselect any existing objects, they will be removed from the selection.

7 Display the Draw toolbar.

8 Click the arrow on the right side of the Fill Color icon  on the Draw toolbar to display the drop-down Color Palette, then click on the light-gray colored square. This will fill the shape components with light gray.

9 Press the Ctrl key and use the down arrow key to move the selection down until it is flush with the bottom of the header band.

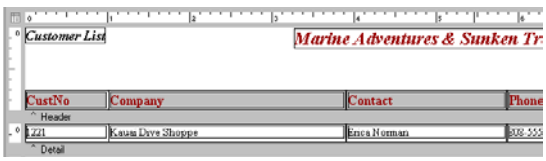
10 Place the mouse over the gray area entitled '^ Detail' and click the right mouse button. Select the Position... menu option. Set the height to 0.2292.

Note: The shapes in the detail band have their ParentHeight property set to True, and thus adjust automatically.

11 Select File | Save from the Delphi main menu.

12 Click the Preview tab in the Report Designer.

layout check



Add a TimeStamp

1 Return to the Design workspace.

2 Click the SystemVariable component icon on the Report Designer component palette.

3 Click on the far left side of the footer band. A System Variable component will be created.

4 Position the component at the top left corner of the footer band.

5 Locate the drop-down list in the upper left corner of the Report Designer. It should contain a list of variable types for the component.

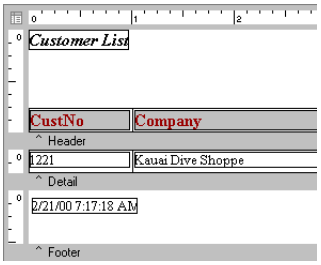
6 Select the PrintDateTime item from this list. The current date and time should be displayed in the component.

7 Set the font:

Font Name	Times New Roman
Font Size	10
Font Color	Black
Font Style	Regular

Note: The PrintDateTime variable type causes the SystemVariable to display the date and time when the report started printing. The same date and time is printed on every page - it is essentially a time stamp that can be used to track the moment the report was printed.

layout check



Add Page Numbers

1 Scroll right until the edge of the footer band is visible.

2 Click the SystemVariable component icon on the Report Designer component palette.

3 Click on the far right side of the footer band. A SystemVariable component will be created.

4 Use the drop-down list on the left side of the Report Designer to select the PageSetDesc variable type. Page 1 of 1 should be displayed in the component.

5 Position the component at the top right corner of the footer band.

6 Right-justify the component by clicking on the right-justify icon on the Format toolbar.

7 Scroll to the left, hold down the Shift key, and select the PrintDateTime System Variable.

8 Click the Align Top icon on the Align or Space toolbar.

9 Click the Select Report icon, which appears at upper left corner of the Report Designer where the horizontal and vertical rulers meet. This will select the report component in the Object Inspector.

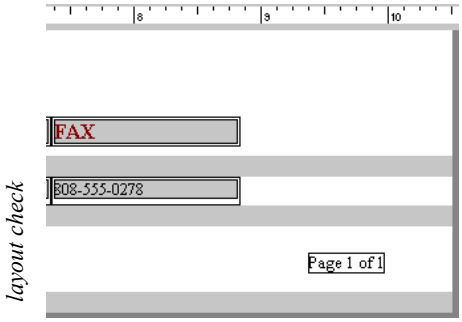
10 Locate the PassSetting property of the Report in the Object Inspector.

11 Make sure it is set to psTwoPass. This property is set automatically when you select a SystemVariable type that requires a Page Count.

Note: Setting the report PassSetting on psTwoPass will cause the report engine to generate all of the pages automatically. One-pass reports only generate the first page, and then any pages requested after that. This does not allow the report to calculate the total number of pages. Two-pass reports are required whenever you are using the PageCount in a System Variable.

12 Select File | Save from the Delphi main menu.

13 Click the Preview tab in the Report Designer.



Preview the Report at Run-time

1 Close the Report Designer window.

2 Select the Standard tab of the Delphi component palette.

3 Add a Button component to the form.

4 Configure the Button component:

Name	btnPreview
Caption	Preview

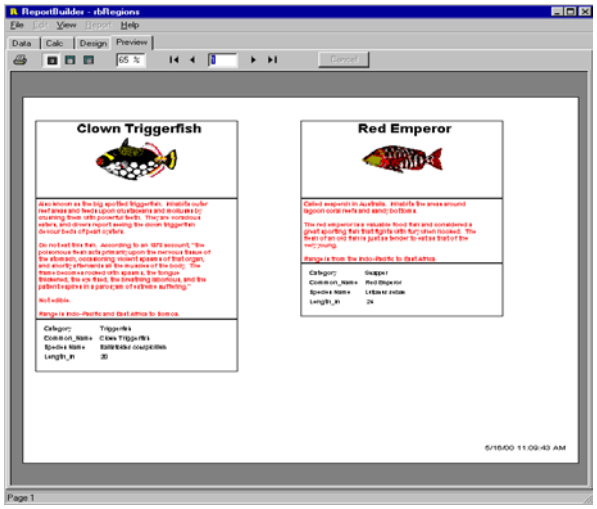
5 Add the following code to the OnClick event handler of the button:

```
rbCustomerList.Print;
```

6 Select File | Save from the Delphi main menu.

7 Run the Project.

8 Click on the Preview button. The DeviceType property of the report component defaults to Screen; thus, the report automatically displays the Print Preview form when the Print method is called. The Preview form should look like this:



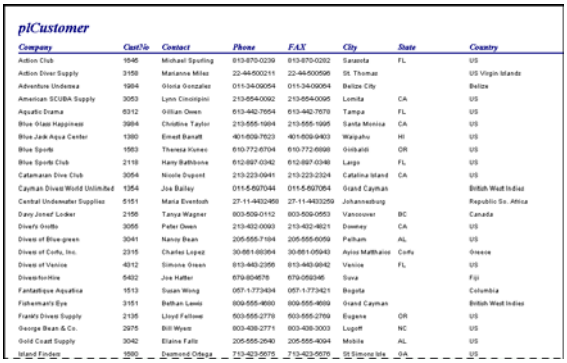
Creating a Report Via the Report Wizard

Overview

This tutorial will show you how to do the following:

- Configure Delphi data access objects for use by reports
- Use the Report Wizard to create report layouts
- Save and load report layouts to report
- template files (.rtm)
- Add code to preview and print the report at runtime

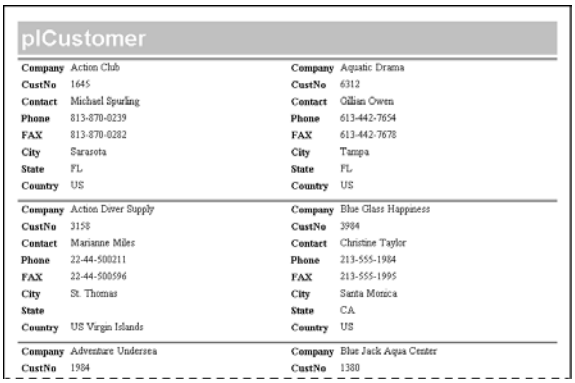
Tabular Report



plCustomer

Company	CustNo	Contact	Phone	FAX	City	State	Country
Action Club	1645	Michael Spurling	813-870-0239	813-870-0282	Sarasota	FL	US
Action Diver Supply	3158	Marianne Miles	22-44-500211	22-44-500596	St. Thomas		US Virgin Islands
Adventure Undersea	1984	Gloria Gonzalez	011-34-09054	011-34-09064	Belize City		Belize
American SCUBA Supply	3053	Lynn Cinciripini	213-654-0092	213-654-0095	Lomita	CA	US
Aquatic Drama	6312	Gillian Owen	613-442-7654	613-442-7678	Tampa	FL	US
Blue Glass Happiness	3984	Christine Taylor	213-555-1984	213-555-1995	Santa Monica	CA	US
Blue Jack Aqua Center	1380	Ernest Barratt	401-609-7823	401-609-9403	Waipahu	HI	US
Blue Sports	1563	Theresa Kovac	610-772-6704	610-772-6698	Gnibaldi	OR	US
Blue Sports Club	2118	Hany Bathhorne	612-897-0342	612-897-0348	Largo	FL	US
Catamaran Dive Club	3064	Nicole Dupont	213-223-0941	213-223-2324	Catalina Island	CA	US
Cayman Divers World Unlimited	1354	Joe Bailey	011-5-607044	011-5-607064	Grand Cayman		British West Indies
Central Underwater Supplies	5151	Maria Eventouh	27-11-4402468	27-11-4403259	Johannesburg		Republic So. Africa
Davy Jones' Locker	2156	Tanya Wagner	603-509-0112	603-509-0553	Vancouver	BC	Canada
Diver's Grotto	3065	Peter Owen	213-402-0093	213-402-4621	Downey	CA	US
Divers of Blue-green	3041	Nancy Bean	205-555-7184	205-555-6059	Pelham	AL	US
Divers of Corfu, Inc.	2315	Charles Lopez	30-661-66364	30-661-05943	Ayios Matthaios	Corfu	Greece
Divers of Venice	4312	Simone Green	613-443-2356	613-443-9642	Venice	FL	US
Divers for Hire	5402	Joe Hafter	679-804676	679-059346	Suva		Fiji
Fantastique Aquatica	1513	Susan Wong	057-1-773434	057-1-773421	Bogota		Columbia
Fisherman's Eye	3151	Bethan Lewis	609-555-4669	609-555-4699	Grand Cayman		British West Indies
Frank's Dives Supply	2135	Lloyd Fellowe	503-555-2779	503-555-2769	Eugene	OR	US
George Bean & Co.	2975	Bill Wyen	803-438-2771	803-438-3003	Lugoff	NC	US
Gold Coast Supply	3042	Elaine Falls	205-555-2640	205-555-4094	Mobile	AL	US
Island Finders	1590	Desmond Odega	713-423-5675	713-423-5678	St Simons Isle	GA	US

Vertical Report



plCustomer

Company	Action Club	**Company**	Aquatic Drama
CustNo	1645	**CustNo**	6312
Contact	Michael Spurling	**Contact**	Gillian Owen
Phone	813-870-0239	**Phone**	613-442-7654
FAX	813-870-0282	**FAX**	613-442-7678
City	Sarasota	**City**	Tampa
State	FL	**State**	FL
Country	US	**Country**	US
Company	Action Diver Supply	**Company**	Blue Glass Happiness
CustNo	3158	**CustNo**	3984
Contact	Marianne Miles	**Contact**	Christine Taylor
Phone	22-44-500211	**Phone**	213-555-1984
FAX	22-44-500596	**FAX**	213-555-1995
City	St. Thomas	**City**	Santa Monica
State		**State**	CA
Country	US Virgin Islands	**Country**	US
Company	Adventure Undersea	**Company**	Blue Jack Aqua Center
CustNo	1984	**CustNo**	1380

Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmViaReportWizard'.

3 Select File | Save As from the Delphi menu and save the form under the name rbViaWiz in the My RB Tutorials directory (established on page 181).

Note: It is important to name the form and save the form's unit using the names given above because the report will be used in later tutorials.

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbRWProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create a Table, DataSource, and DataPipeline Component

1 Select the Data Access tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblCustomer
TableName	customer.db
IndexFieldName	Company

4 Add a DataSource component to the form.

5 Configure the DataSource component:

DataSet	tblCustomer
Name	dsCustomer

6 Select the RBuilder tab of the Delphi component palette.

7 Add a DBPipeline component to the form.

8 Configure the DBPipeline component:

DataSourcedsCustomer

NameplCustomer

Create a Report and Connect it to the Data

1 Add a Report component to the form.

2 Configure the Report component:

DataPipeline	plCustomer
Name	rbCustomerList

Invoke the Report Designer and Access the Report Wizard

1 Double-click on the Report component to display the Report Designer.

2 Select File | New from the Report Designer's menu.

3 Double-click the Report Wizard icon (or click the OK button while the icon is selected).

Use the Report Wizard to Lay Out a Tabular Style Report

1 Click on the Data Pipeline Name drop-down list and select the Customer table.

2 Click on the Company field in the Available Fields list box and click the arrow button '>' to move the field to the Selected Fields list box.

Note: You can also move a field to the Select fields list box by double-clicking it.

3 Repeat this procedure to select the following fields:

CustNo
Contact
Phone
FAX
City
State
Country

4 Click the Next button to access the next page of the Wizard.

5 This page of the Report Wizard is used to define groups. No groups are required for this example; therefore, click the Next button again.

6 Click the Landscape radio button located in the Orientation group box.

7 Click the Next button.

8 Click on each of the style options in the list box to preview each of the available styles.

9 Select the Corporate style and click the Next button.

10 Click the Finish button. The report layout will be generated and the Preview tab of the Report Designer will be selected so that you can view the report.

11 Click the Design tab.

12 Select the CustNo label and DBText components and click the Left Justify icon.

layout check

Preview the Report

1 Click the Preview tab.

2 Click the Page Width and 100% buttons to view the report on a larger scale.

3 Type 80 in the % edit box and press enter.

4 Type 120 in the edit box and press enter again. The report can be scaled for viewing at any size up to 250%.

5 Click the Page Width button to continue viewing the report.

6 Click the Next and Prior page buttons to navigate through the pages of the report.

7 Click the Print button to access the print dialog. From here you can print All, Current, or a specified Range of pages. Click the Cancel button to close the dialog.

8 Click the Design tab.

Set the Report to Two-Pass Mode

1 Select the System Variable component on the far right of the footer band.

2 Use the drop-down list control in the far left corner of the Report Designer to change the variable type from PageNoDesc to PageSetDesc.

3 Click the Select Report icon, which appears at upper left corner of the Report Designer where the horizontal and vertical rulers meet. This will select the report object in the Object Inspector.

4 Locate the PassSetting property of the Report in the Object Inspector.

5 Make sure it is set to psTwoPass. This property is set automatically when you select a SystemVariable type that requires a Page Count.

Note: Setting the report PassSetting on psTwoPass will cause the report engine to generate all of the pages automatically. One-pass reports only generate the first page, and then any pages requested after that. This does not allow the report to calculate the total number of pages. Two-pass reports are required whenever you are using the PageCount in a System Variable.

Save the Report Layout to a Template File

1 Select File | Save As from the Report Designer's menu.

2 Type CustTab in the File Name edit box and click the Save button. Be sure to save the template in the same directory as your project.

Note: By default, the report layout will be saved with the form on which the TppReport component resides. Saving the layout of a report to a template file is optional. However, for this example we want to create two report layouts for the same data. The user can choose which layout to print at runtime. Therefore, we will save the report layouts to .rtm files and load the appropriate layout at runtime.

Use the Report Wizard to Lay Out a Vertical Style Report

1 Select File | New from the Report Designer's menu.

2 Double-click the ReportWizard icon (or click the OK button while the icon is selected).

3 Click on the Data Pipeline Name combo box and select the Customer table.

4 Click on the Company row of the Available Fields list box and click the arrow button '>' to move the field to the Selected Fields list box.

5 Repeat this procedure to select the following fields:

CustNo
Contact
Phone
FAX
City
State
Country

6 Click the Next button to access the next page of the Wizard.

7 This page of the Report Wizard is used to define groups. For this example no groups are required; therefore, click the Next button again.

8 Click the Vertical radio button located in the Layout group box.

9 Click the Next button.

10 Select the Soft Gray style and click the Next button.

11 Click the Finish button. The report layout will be generated and the Preview tab of the Report Designer will be selected so that you can view the report layout.

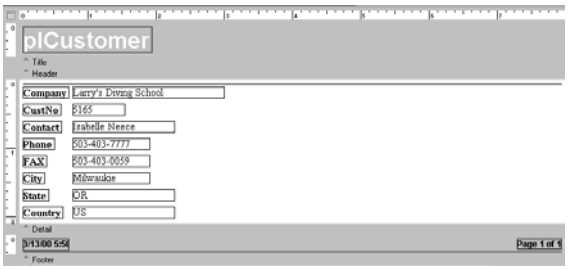
12 Click the Design tab.

13 Select the CustNo label and DBText components and click the Left Justify icon.

14 Select the System Variable component on far right of the footer band.

15 Use the drop-down list control in the far left corner of the Report Designer to change the variable type from PageNoDesc to PageSetDesc.

layout check



Modify the Report Layout to Contain Columns

1 Select File | Page Setup from the Report Designer main menu.

2 Select the Layout tab.

3 Click the Up Arrow button located to the right of the Columns edit box. This should modify the number of columns from 1 to 2.

4 Click the OK button.

5 Right-click over the line component at the top of the detail band and set the ParentWidth property to True.

Note: When using the Component popup menus, a checkmark next to a menu option indicates the property is set to True. Selecting the menu option toggles the property from Checked to Unchecked (i.e. True to False) and vice-versa.

6 Preview the report again. You should see two columns.

Note: Quite often, you will want to use the Report Wizard to layout a report and then customize the report layout further using the Report Designer.

Save the Report Layout to a Template File

1 Return to the design workspace.

2 Select File | Save As from the Report Designer main menu.

3 Type CustVert in the File Name edit box and click the Save button. Be sure to save the template in the My RB Tutorials directory.

Note: You can use the File | Open menu to access the file open dialog and load the CustTab.rtm and CustVert.rtm files at design-time. To save any changes, use the File | Save menu option.

Add the Run-Time Interface Components

1 Close the Report Designer by selecting File | Close from the Report Designer main menu.

Note: You can also close the Report Designer by clicking the X button located on the upper right corner of the Window Title bar or by double-clicking on the R icon  located in the upper left corner of the Window Title bar.

2 Select the Standard tab of the Delphi component palette.

3 Add a Group Box component to the form.

4 Configure the Group Box component:

Name	gbReportStyle
Caption	Report Style
Left	15
Top	11
Width	131
Height	82

5 Add two Radio Button components to the Group Box:

Name	rbTabular
Caption	Tabular
Checked	True
Left	18
Top	23
Width	65

Name	rbVertical
Caption	Vertical
Left	18
Top	50
Width	65

6 Add two Button components to the form.

7 Configure the Button components:

Name	btnPreview
Caption	Preview
Left	163
Top	28
Name	btnPrint
Caption	Print
Left	163
Top	56

Code the Event Handlers

1 In the private section of the form unit, add the following declaration:

```
private
   FPathName: String;
```

2 Add an event handler for the Form OnCreate event:

Code — *Event handler for the Form OnCreate event*

```
procedure TfrmViaReportWizard.FormCreate(Sender: TObject);
begin
   {get the path name for this application}
   FPathName := ExtractFilePath(ParamStr(0));

   {set the initial report template file name}
   rbCustomerList.Template.FileName := FPathName + 'CustTab.rtm';

end;
```

3 Add event handlers for each Button's OnClick event:

Code — *Event Handler for the Button's OnClick event*

```
procedure TfrmViaReportWizard.btnPreviewClick(Sender: TObject);
begin
   {load the template file and print to screen}
   rbCustomerList.Template.LoadFromFile;
   rbCustomerList.DeviceType := dtScreen;
   rbCustomerList.Print;
end;

procedure TfrmViaReportWizard.btnPrintClick(Sender: TObject);
begin
   {load the template file and print to printer}
   rbCustomerList.Template.LoadFromFile;
   rbCustomerList.DeviceType := dtPrinter;
   rbCustomerList.Print;
end;
```

4 Add event handlers for each Radio Button's OnClick event as shown below.

5 Add ppTypes to the uses clause of your form's unit.

Note: The ppTypes unit contains the declaration for all enumerated types and constants used in ReportBuilder. In this instance, it is needed to support the references to dtPrinter and dtScreen in the preceding button click event handlers.

Compile and Run the Application

1 Select Project | Compile | rbRWProj. Fix any compilation problems.

2 Select File | Save from the Delphi main menu.

3 Close the form and the unit.

4 Run the project.

5 Click the Preview button to view the tabular style report.

6 Click the Vertical radio button.

7 Click the Preview button to view the vertical style report.

Note: You can also send the report to the printer by clicking the Print button.

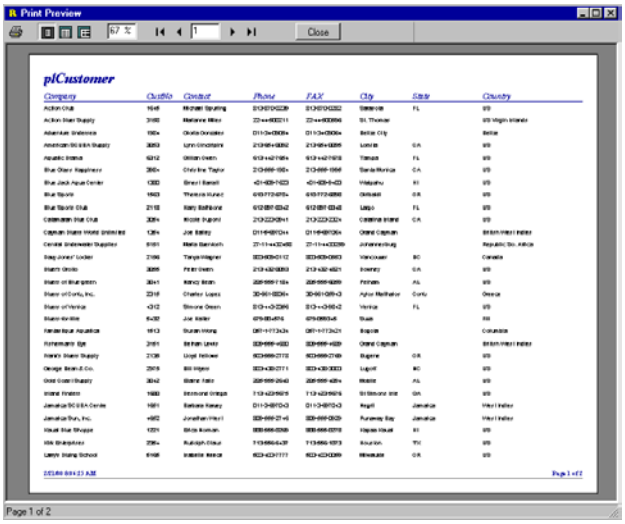
.

Code *Event handler for Radio Buttons' OnClick*

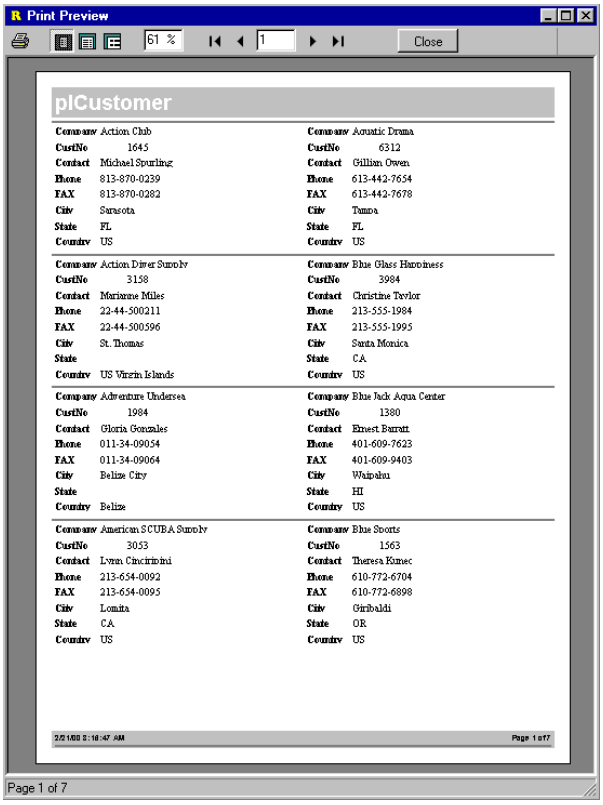
```
procedure TfrmViaReportWizard.rbTabularClick(Sender: TObject);
begin
   {set template file name for the tabular report}
   rbCustomerList.Template.FileName := FPathName + 'CustTab.rtm';
end;
procedure TfrmViaReportWizard.rbVerticalClick(Sender: TObject);
begin
   {set template file name for the vertical report}
   rbCustomerList.Template.FileName := FPathName + 'CustVert.rtm';
end;
```

The reports should appear as pictured below:

Tabular Report



Vertical Report



A Simple Report the Hard Way

Overview

This tutorial will show you how to do the following:

- Configure Delphi data access objects for use by reports
- Work with the Report Designer
- Create and configure the most common report components

Customer List

Marine Adventures and Sunken Treasures

Company	Contact	Phone	Address	City	State
Kauai Dive Shoppe	Erica Norman	808-555-0269	4-976 Sugarloaf Hwy	Kapaa Kauai	HI
Unisco	George Weathers	809-555-3915	PO Box Z-547	Freeport	
Sight Diver	Phyllis Spooner	357-6-876708	1 Neptune Lane	Kato Paphos	
Cayman Divers World Unlimited	Joe Bailey	011-3-697044	PO Box 541	Grand Cayman	
Tom Sawyer Diving Centre	Chris Thomas	504-798-3022	632-1 Third Fryderskoj	Christiansted	St. Croix
Blue Jack Aqua Center	Ernest Barrett	401-609-7623	23-738 Paddington Lane	Waipahu	HI
VIP Divers Club	Russell Christopher	809-453-5976	32 Main St.	Christiansted	St. Croix
Ocean Paradise	Feril Gardner	808-555-8231	PO Box 8745	Kailua-Kona	HI
Fantastique Aquatica	Susan Wong	057-1-773434	Z32 999 #12A-77 A.A.	Bogota	
Marriott Divers Club	Joyce Marsh	416-698-0399	872 Queen St.	Kitchener	Ontario
The Depth Charge	Sam Witherspoon	800-555-3798	15243 Underwater Fwy.	Marathon	FL
Blue Sports	Theresa Kunec	610-772-6704	203 12th Ave. Box 746	Garibaldi	OR
Makai SCUBA Club	Donna Sims	317-649-9098	PO Box 8534	Kailua-Kona	HI
Action Club	Michael Spurling	813-870-0239	PO Box 5451-F	Sarasota	FL
Jamaica SCUBA Centre	Barbara Harvey	011-3-697043	PO Box 68	Negril	Jamaica
Island Finders	Desmond Ortega	713-423-5675	6133 1/3 Stone Avenue	St Simons Isle	GA
Adventure Undersea	Gloria Gonzales	011-34-09054	PO Box 744	Belize City	
Blue Sports Club	Harry Bathbone	612-897-0342	63363 Nez Perce Street	Largo	FL
Frank's Divers Supply	Lloyd Fellows	503-555-2778	1455 North 44th St.	Eugene	OR

Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmCustomerList'.

3 Select File | Save As from the Delphi menu and save the form under the name rbCust in the My RB Tutorials directory (established on page 181).

Note: It is important to name the form and save the form's unit using the names given above because the report will be used in later tutorials.

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbCSProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create a Table, DataSource, and DataPipeline Component

1 Select the BDE tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblCustomer
TableName	customer.db

4 Add a DataSource component to the form.

5 Configure the DataSource component:

DataSet	tblCustomer
Name	dsCustomer

6 Select the RBuilder tab of the Delphi component palette.

7 Add a DBPipeline component  to the form.

8 Configure the DBPipeline component:

DataSource	dsCustomer
Name	plCustomer

Create a Report and Connect it to the Data

1 Add a Report component  to the form.

2 Configure the Report component:

DataPipeline	plCustomer
Name	rbCustomerList

Invoke the Report Designer and Set the Paper Orientation

1 Double-click on the Report component to display the Report Designer.

2 Size and move the Report Designer window so that the Object Inspector is visible.

3 Select the File | Page Setup option from the Report Designer main menu.

4 Click the Paper Size tab and change the paper orientation from Portrait to Landscape, then click the OK button.

Lay Out the Customer List Title Label in the Header Band

1 Right-click over the white space of the header band and access the Position... dialog.

2 Set the height to 0.9167 and click the OK button. The header band should expand in height.

Note: You can also change the height of the header band by dragging the divider that appears below the white space of the header band. This method is quick and easy, but not as precise.

3 Click the Label component icon  on the Report Designer component palette.

4 Click on the left side of the header band. A label will be created in the header band. It will become the current selection in the Object Inspector.

5 Locate the Edit box at the upper left corner of the Report Designer.

6 Select the text inside it and replace it with Customer List.

7 Locate the font controls on the Format bar.

Note: The font controls appear to the right of the Edit box and provide the following functions:

- Font Name drop-down list
- Font Size drop-down list
- Bold
- Italic
- Underline Font Style
- Text Alignment
- Font color
- Highlight color

8 Set the font:

Font Name	Times New Roman
Font Size	12
Font Style	Bold & Italic

9 Locate the horizontal ruler. It is above the header band.

10 Position the mouse over the ruler and right-click.

11 Select the Millimeters option from the speed menu.

12 Right-click over the label and select the Position... menu option.

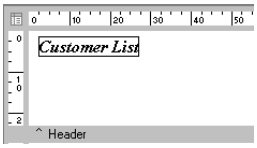
13 Enter the following values into the Position dialog:

Left	2.381
Top	1.323
Width	24.871
Height	5.027

14 Click the OK button. The label should move to the upper left corner of the header band.

Note: You can also change the position of a component by dragging it. The exact position of the component will be displayed in the status bar at the bottom of the Report Designer after it is dropped. You can use any unit of measure (inches, screen pixels, millimeters, etc.) you prefer.

layout check



Lay Out the Company Title Label in the Header Band

1 Place another label near the center of the header band.

2 Configure the label:

Caption	Marine Adventures & Sunken Treasures Co.
Font Name	Times New Roman
Font Size	16
Font Style	Bold & Italic

3 Click the font color drop-down icon and select dark blue.

4 Select View | Toolbars from the Report Designer main menu. A list of toolbar names should be displayed. Select the Align or Space toolbar.

5 Click on the label component that you just created in the header band.

6 Click the Center Horizontally icon on the Align or Space toolbar. This should cause the label to be centered in the header band.

7 Click on the Customer List label.

8 Hold down the Shift key and click the Report Title label.

9 Click the Align Top icon of the Align or Space toolbar. The title labels should now be aligned.

layout check

Create Labels for the Header Band

1 Place six labels in the header band and position them from left to right. Set the label captions as follows:

Company
Contact
Phone
Address
City
State

2 Click on the Company label.

3 Hold down the Shift key and click on the Contact label.

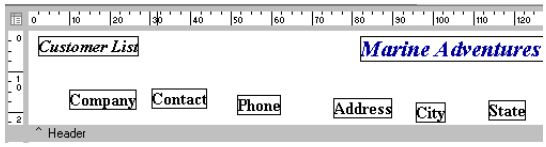
4 Repeat this procedure until all six labels are selected.

5 Set the font:

Font Name	Times New Roman
Font Size	12
Font Style	Bold
Font Color	Black

6 Leave these labels at their current positions; we will align them later.

layout check



Complete the Header Band Layout

1 Click the Line component icon on the Report Designer component palette.

2 Click in the header band. A line component will be created.

3 Right-click over the line and select the Parent-Width menu option. The line will resize to the width of the header band.

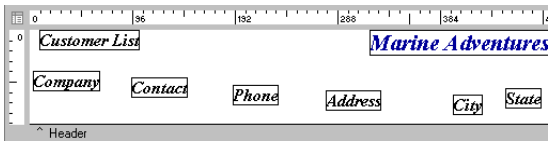
4 Select the View | Toolbars menu option from the Report Designer main menu and make the Draw toolbar visible.

5 Drag the Draw toolbar to the left side of the Report Designer and dock it below the Align or Space toolbar.

6 Click the Line Thickness icon on the Draw toolbar and select the 1 pt thickness. The line thickness should adjust accordingly.

7 Hold down the Ctrl key and press the down arrow key until the line is positioned at the bottom of the header band.

layout check



Lay Out the Detail Band

1 Click the DBText component icon on the Report Designer component palette.

2 Click in the detail band. A DBText component will be created.

3 Locate the two drop-down lists at the upper left corner of the Report Designer. The list on the left contains the data pipelines. It should default to the Customer data pipeline, since this is the data pipeline for the report. The list on the right contains a list of fields for the currently selected data pipeline.

4 Select the Company field from the field list. A company name should be displayed in the DBText component.

5 Set the DBText component's font:

Font Name	Times New Roman
Font Size	10
Font Style	Regular
Font Color	Black

6 Place five more DBText components in the detail band. Assign them to the following fields:

Contact
Phone
Addr1
City
State

7 Right-click over the horizontal ruler and change the unit of measure to Screen Pixels.

8 Right-click over the DBText component assigned to the Company field and access the Position... dialog. Set the position values:

Left	2
Top	4

9 Access the Position... dialog of each remaining DBText component and set the left position to the following values:

Contact	219
Phone	378
Address	519
City	739
State	896

10 Select the Company DBText component, then Shift-click to select the remaining DBText components.

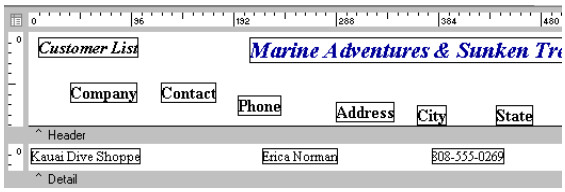
11 Click on the Align Top icon of the Align or Space toolbar. The components should align with Company DBText component.

12 Right-click over each of the components and set Autosize to True.

13 Right-click over the white space of the detail band and access the Position... dialog.

14 Set the height to 25. The detail band should shrink in height.

layout check



Align the Header Band Labels Vertically

1 Select the Company label and right-click over it to access the Position... dialog.

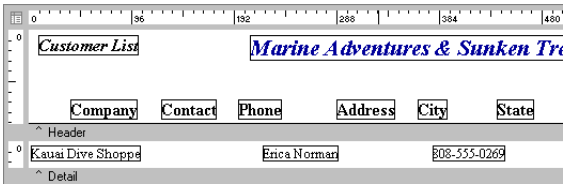
2 Set the Top position to 65.

3 Hold down the Shift key and select the remaining label components in the header band:

Contact
Phone
Address
City
State

4 Click the Align Top icon on the Align or Space toolbar. The labels should align with the top of the Company label.

layout check



Align the Header Band Labels with the Detail Band DBText Components

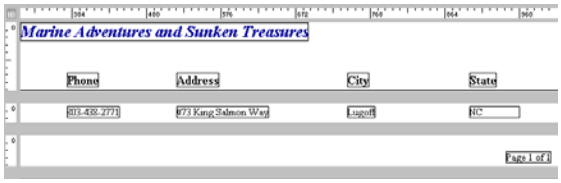
1 Select the Company DBText component in the detail band.

2 Hold down the Shift key and click on the Company label component in the header band.

3 Click the Align Left icon on the Align or Space toolbar. The label and DBText component should now be aligned.

4 Repeat this procedure for the remaining DBText components and corresponding Label components. Make sure to select the DBText component first. The first component selected determines the position used for alignment.

layout check



Set PrintDateTime

1 Right-click over the white space of the footer band and access the Position... dialog.

2 Set the height to 40. The footer band should shrink in height.

3 Click the SystemVariable component icon on the Report Designer component palette.

4 Click on the far left side of the footer band. A System Variable component will be created.

5 Locate the drop-down list on the left-hand side of the Report Designer. It should contain a list of variable types for the component.

6 Select the PrintDateTime item from this list. The current date and time will be displayed in the component.

7 Set the font:

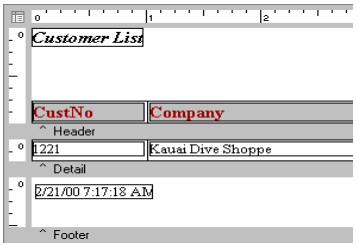
Font Name	Times New Roman
Font Size	10
Font Style	Regular
Font Color	Black

8 Right-click over the SystemVariable component and access the Position... dialog.

9 Set the position values:

Left	9
Top	22

layout check



Set Page Numbers

1 Locate the horizontal scroll bar at the bottom of the Report Designer.

2 Scroll right until the right edge of the footer band is visible.

3 Click the SystemVariable component icon on the Report Designer component palette.

4 Click on the far right side of the footer band. A SystemVariable component should be created.

5 Use the drop-down list on the left-hand side of the Report Designer to select the PageSetDesc variable type. Page 1 of 1 should be displayed in the component.

6 Right-click over the component and access the Position... dialog.

7 Set the position values:

Left	943
Top	22

8 Right-justify the component by clicking on the Right Justify icon on the Format toolbar.

9 Select File | Save from the Delphi main menu.

10 Click the Preview tab in the Report Designer. A three page report should appear.

layout check

Preview the Report at Run-Time

1 Close the Report Designer.

2 Select the Standard tab of the Delphi component palette.

3 Add a Button component to the form.

4 Configure the Button component:

Name	btnPreview
Caption	Preview

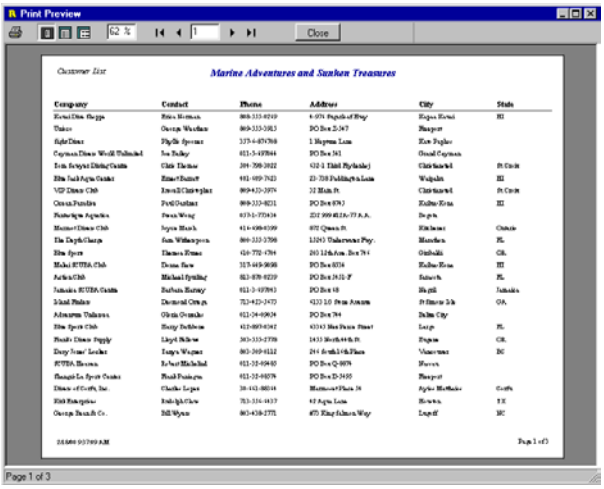
5 Add the following code to the OnClick event handler of the button:

```
rbCustomerList.Print;
```

6 Select File | Save from the Delphi main menu.

7 Run the Project.

8 Click on the Preview button. The report will be displayed in the Print Preview form:

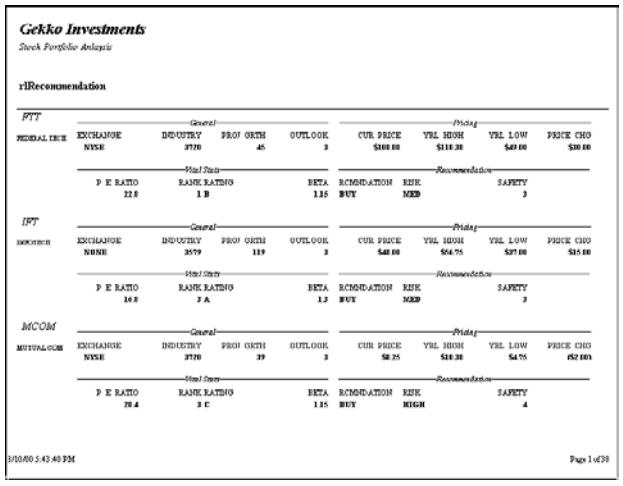


Groups, Calculations, and the Summary Band

Overview

This tutorial will show you how to do the following:

- Divide a report into sections using groups
- Perform calculations within a report
- Utilize a summary band



Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmStockSummary'.

3 Select File | Save As from the Delphi menu and save the form under the name rbStock in the My RB Tutorials directory (established on page 181).

Note: It is important to name the form and save the form's unit using the names given above because the report will be used in later tutorials.

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbSTProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create a Query, DataSource, and DataPipeline Component

1 Select the Data Access tab of the Delphi component palette.

2 Add a Query component to the form.

3 Configure the Query component:

DatabaseName	DBDEMOS
SQL	SELECT * FROM master ORDER BY rcmndation
Name	qryStock

4 Double-click on the Active property in the Object Inspector to set it to true.

5 Add a DataSource component to the form.

6 Configure the DataSource component:

DataSet	qryStock
Name	dsStock

7 Select the RBuilder tab of the Delphi component palette.

8 Add a DBPipeline component  to the form.

9 Configure the DBPipeline component:

DataSource	dsStock
Name	plStock

Create a Report and Connect it to the Data

1 Add a Report component  to the form.

2 Configure the Report component:

DataPipeline	plStock
Name	rbStock

Invoke the Report Designer

1 Double-click on the Report component to display the Report Designer.

2 Size and move the Report Designer window so that the Object Inspector is visible.

3 Set the paper orientation to Landscape (File | Page Setup | Paper Size).

4 Drag the header band to the one inch mark on the vertical ruler. Right-click over the white space of the header band and access the Position dialog. The value for the height should be 1.

Create Labels in the Header Band

1 Place a label in the left side of the header band.

2 Set the caption to Gekko Investments.

3 Set the font and position:

Font Name	Times New Roman
Font Size	20
Font Style	Bold & Italic
Left	0.0417
Top	0.0417

4 Select the label.

5 Copy the label.

6 Paste the label into the header band.

7 Set the caption of this label to Stock Portfolio Analysis.

8 Set the font and position:

Font Name	Times New Roman
Font Size	12
Font Style	Bold & Italic
Left	0.0417
Top	0.4063

layout check

Create a Group on the 'rcmndation' Field

1 Select Report | Groups from the main menu. The Groups dialog will be displayed.

2 The drop-down list at the top of the dialog contains a list of fields from the Query. Select the plStock.RCMNDATION field and click the Add button. A new entry entitled Group1: plStock.RCMNDATION should be added to the list.

3 Locate the Start New Page check box near the middle of the dialog.

4 Check this box. This will cause the report to start a new page each time the RCMNDATION field changes value.

Note: Because groups are based on a change in field value, the field on which a group is based should always be used to order the records. You may remember that the query for this report is ordered by the 'rcmndation' field. You must always order the records by the fields you intend to use for groups because ReportBuilder will not sort records for you.

5 Click the OK button. You should see two new bands: one above and one below the detail band. These bands have no height; thus, no white space appears above the dividers.

6 Right-click over the group header band divider and select the Position... menu option.

7 Set the height to 0.6458.

layout check

Lay Out the Group Header Band

1 Place a label in the left side of the group header band.

2 In the Object Inspector, set the Name of the component to rlRecommendation.

3 Set the font and position:

Font Name	Times New Roman
Font Size	14
Font Style	Bold
Left	0.0417
Top	0.0937

4 Place a line component near the bottom of the group header band.

5 Right-click over the line component and select the ParentWidth option.

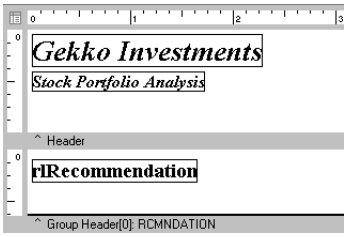
6 Locate the Line type drop-down list at the upper left corner of the Report Designer.

7 Select 'Bottom' from this list. This causes the line to be drawn at the bottom of the rectangular bounding shape.

8 Click on the Line thickness icon on the Draw toolbar and select 1 1/2 point.

9 Select the line and use the arrow keys to position it so that it's flush with the bottom of the group header band.

layout check



Code the BeforeGenerate Event of the Group Header Band

1 Click in the white space of the group header band. The band should be selected in the Object Inspector.

2 Click on the Events tab of the Object Inspector and double-click on the BeforeGenerate event. Add the code shown below to the event handler.

Note: This routine retrieves the value of the recommendation field into a local string variable. It then assigns a color based on the value of the field. The color is stored in a private variable of the form so that all of the report event handlers can use it. The recommendation label in the group header band is set to the color. Finally, the recommendation label caption is set using the FirstPage property of the group. If FirstPage is False, then the group has overflowed onto an additional page.

Code | *Event handler for the BeforeGenerate Event*

```
procedure TfrmStockSummary.ppGroupHeaderBand1BeforeGenerate(Sender: TObject);
var
   lsRecommendation: String;
begin
   lsRecommendation := plStock['Rcmndation'];

   {determine the font color based on the value of Recommendation}
   if (lsRecommendation = 'BUY') then
     FRecommendationColor := clGreen

   else if (lsRecommendation = 'HOLD') then
     FRecommendationColor := clOlive

   else {lsRecommendation = 'SELL'}
     FRecommendationColor := clMaroon;

   rlRecommendation.Font.Color := FRecommendationColor;

   {check whether first page of group or a continuation}
   if rbStock.Groups[0].FirstPage then
     rlRecommendation.Caption := 'Recommended ' + lsRecommendation + ' List'
   else
     rlRecommendation.Caption := 'Recommended ' + lsRecommendation + '''s continued...';

end;
```

3 Scroll up in the Code Editor and add the following code to the 'private' section of the form class declaration:

```
FRecommendationColor: TColor;
```

4 Select View | Toggle Form/Unit from the Delphi main menu.

5 Select the Standard tab of the Delphi Component palette.

6 Create a button on the form.

7 Configure the button:

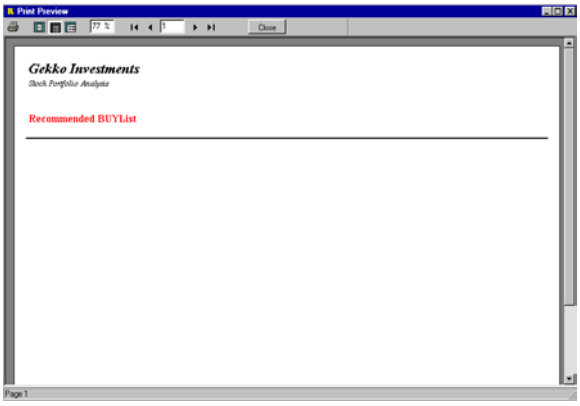
Name	btnPreview
Caption	Preview

8 Add the following code to the OnClick event for the button:

```
rbStock.Print;
```

9 Select File | Save from the Delphi main menu.

10 Run the project. The report should look like this:



Add General Data to the Detail Band

1 Access the Report Designer.

2 Right-click over the white space of the detail band and select the Position... menu option.

3 Set the height to 1.8542.

4 Select View | Toolbars | Data Tree. The Data Tree window should appear. Position it over the Object Inspector.

5 Click the Layout tab at the bottom of the Data Tree and configure the drag-and-drop settings:

Create	All
Style	Tabular
Label Grid	False
Label Font Name	Times New
Label Font Style	Regular
Label Font Size	10 point
Label Font Color	Black
Field Grid	False
Field Font Name	Times New
Field Font Style	Bold
Field Font Size	10 point
Field Font Color	Black

6 Click on the Data tab and use the Ctrl-click method to select these fields in order:

EXCHANGE
INDUSTRY
PROJ_GRTH
OUTLOOK

7 Drag the selected fields into the detail band. Four DBText components and four associated labels will be created.

8 Hold down the Ctrl key and use the arrow keys to position the selection:

Left	1.0729
Top	0.3542

Note: As you move the selection, the position appears on the status bar. Use these measurements to determine when you have reached the position.

9 Place a line above the selection.

10 Configure the line component:

Line Type	Top
Thickness	1 1/2 pt
Left Position	1.0729
Top	0.2083
Width	4.2604

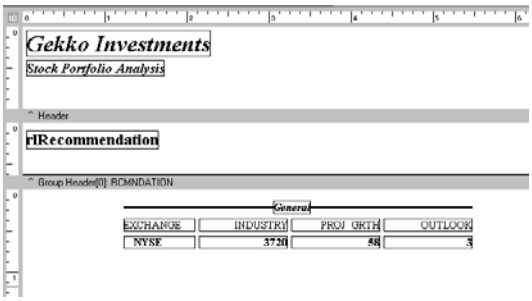
11 Place a label component over the line.

12 Configure the label component:

Caption	General
Font Name	Times New Roman
Font Size	10
Font Style	Bold & Italic
Text Alignment	Centered
Top Position	0.1354
Transparent	False

13 Select the line, then Shift-click the label. Click the Align Middle icon on the Align or Space toolbar.

layout check



Add the Vital Stats Data to the Detail Band

1 Use the Ctrl-click method to select these fields in order from the Data Tree:

P_E_RATIO
RANK
RATING
BETA

2 Drag the selected fields directly below the General components. Four DBText components and four associated labels will be created.

3 Hold down the Ctrl key and use the arrow keys to position the selection:

Left	1.0729
Top	1.1875

4 Select the line and General label.

5 Copy and paste the selection. A new line and label should appear just below the originals.

6 Position these two components above the newly-created components.

7 Set the line top to 1.0417.

8 Set the label top to 0.9687.

9 Right-click over the label and select the Bring to Front menu item.

10 Set the label caption to Vital Stats.

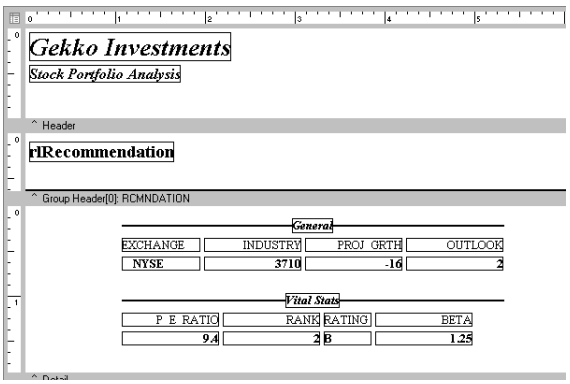
11 Select the PE RATIO label, then Shift-click the line.

12 Click the Align Left icon of the Align or Space toolbar.

13 Select the line, then Shift-click the Vital Stats label.

14 Click on the Align Middle icon of the Align or Space toolbar.

layout check



Add the Pricing Data to the Detail Band

1 Use the Ctrl-click method to select these fields in order from the Data Tree:

CUR_PRICE
YRL_HIGH
YRL_LOW
PRICE_CHG

2 Drag the selected fields to the immediate right of the General components. Four DBText components and four associated labels will be created.

3 Hold down the Ctrl key and use the arrow keys to position the selection:

Left	5.7188
Top	0.3542

4 Right-click over each of the four DBText components and set the DisplayFormat to $#,0.00;($#,0.00) (the first selection with a dollar sign).

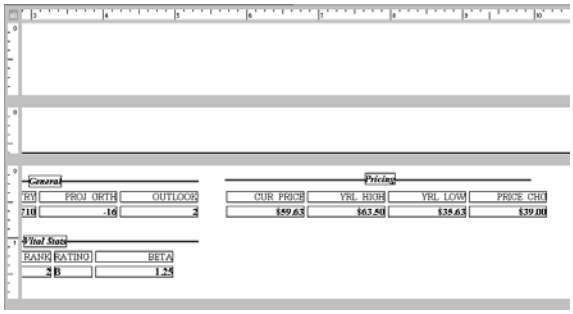
5 Select the line and General label.

6 Copy and paste the selection.

7 Position the selection above the newly-created components.

8 Set the label caption to Pricing.

layout check



Align the Pricing Data Components

1 Select the CUR_PRICE label and then the line component.

2 Click on the Align Left icon of the Align or Space toolbar.

3 Set the width of the line to 4.4479.

4 Select the General label and then the Pricing label.

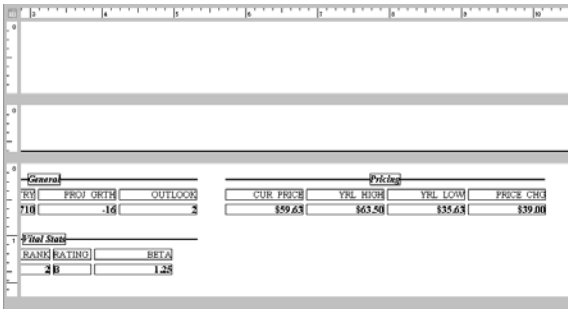
5 Click on the Align Top icon of the Align or Space toolbar.

6 Select the line behind the General label, then the line behind the Pricing label.

7 Click the Align Top icon.

8 Select the Pricing line and then the Pricing label.

9 Click on the Align Middle icon of the Align or Space toolbar. The label should be centered horizontally in relation to the line.



layout check

Add the Recommendation Data to the Detail Band

1 Use the Ctrl-click method to select these fields in order from the Data Tree:

RCMNDATION
RISK
SAFETY

2 Drag the selected fields immediately below the Pricing components.

3 Hold down the Ctrl key and use the arrow keys to position the selection:

Left	5.7188
Top	1.1875

4 Close the Data Tree window.

5 Click in the white space of the band to deselect the components, then select the DBText component for the RCMNDATION field.

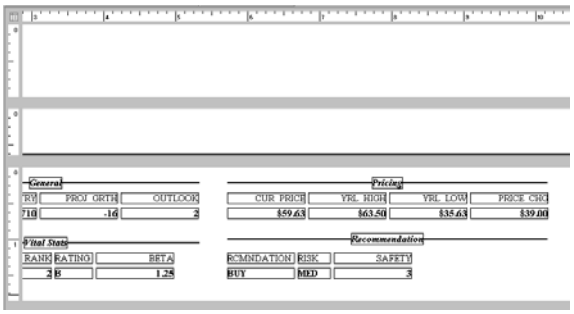
6 Set the name of this component to dbtRecommendation in the Object Inspector. We will use this name in an event handler.

7 Select the Pricing line and label.

8 Copy and paste the selection.

9 Position the selection above the newly-created components.

10 Set the label caption to Recommendation.



layout check

Align the Recommendation Data Components

1 Select the RCMNDATION label and then the line component.

2 Click on the Align Left icon of the Align or Space toolbar.

3 Select the Vital Stats label and then the Recommendation label.

4 Click on the Align Top icon of the Align or Space toolbar.

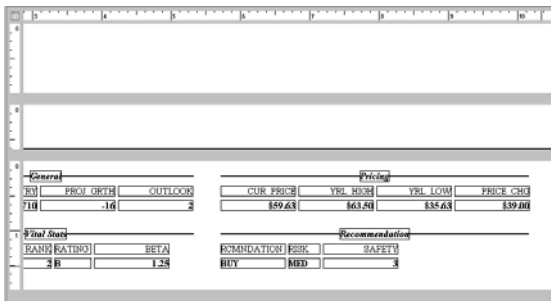
5 Select the line behind the Vital Stats label, then the line behind the Recommendation label.

6 Click the Align Top icon of the Align or Space toolbar.

7 Select the Recommendation line and label.

8 Click on the Align Middle icon of the Align or Space toolbar. The label should be centered horizontally in relation to the line.

layout check



Add the Stock Symbol and Company Data to the Detail Band

1 Place a DBText component in the upper left corner of the detail band

2 Configure the DBText component:

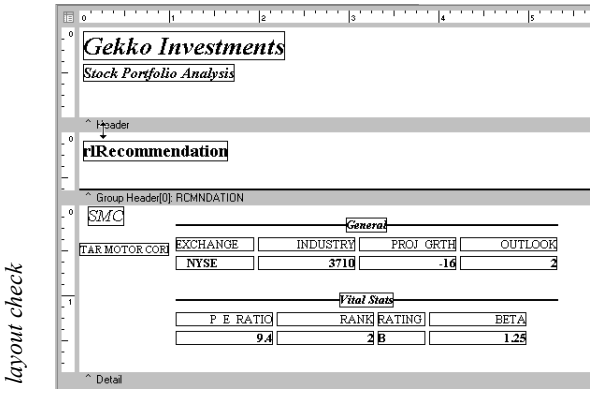
DataField	SYMBOL
AutoSize	True
Name	dbtSymbol
Font Name	Times New Roman
Font Size	14
Font Style	Italic
Text Alignment	Left justified
Left	0.0937
Top	0.0104

3 Place another DBText component in the upper left corner of the detail band.

4 Configure the DBText component:

DataField	CO_NAME
Name	dbtCompany
Font Name	Times New Roman
Font Size	8
Font Style	Regular
Left	0
Top	0.4167
Width	1

layout check



Assign the BeforeGenerate Event Handler of the Detail Band

1 Click in the white space of the detail band.

2 Click on the Events tab of the Object Inspector.

3 Double-click on the BeforeGenerate event. An event handler shell will be generated in your Delphi form.

4 Add the code as shown below.

Code	*Set the Font Color*

```
{set the font color of Symbol and Recommendation}
dbtRecommendation.Font.Color := FRecommendationColor;
dbtSymbol.Font.Color := FRecommendationColor;
```

Note: This routine sets the color of the stock symbol and the recommendation field to the color as defined in the BeforeGenerate event of the group header band.

5 Select File | Save from the Delphi main menu.

6 Select Project | Compile rbSTProj from the Delphi main menu. Fix any problems.

7 Run the project.

8 Preview the report. Click on the Last Page icon of the Print Preview window. A thirty-seven page report should appear. The label in the group header should change color based on the group. The stock symbol and recommendation fields should also be color-coded.

Lay Out the Footer Band

1 Access the Report Designer.

2 Place a System Variable component on left side of the footer band.

3 Configure the SystemVariable component:

VarType	PrintDateTime
Text Alignment	Left justified
Font Name	Times New Roman
Font Size	10
Font Style	Regular
Font Color	Black
Left	0
Top	0.2292

4 Scroll to the right until you see the edge of the footer band.

5 Create a SystemVariable component on the right side of the footer band.

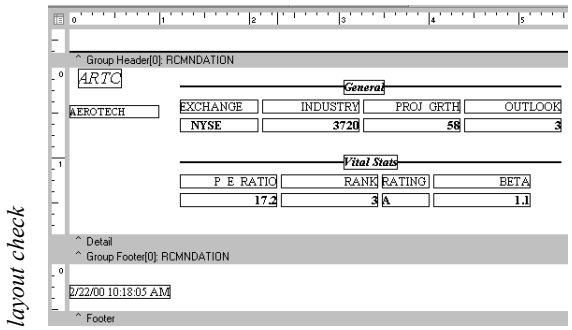
6 Configure the SystemVariable component:

VarType	PageSetDesc
Text Alignment	Right justified
Font Name	Times New Roman
Font Size	10
Font Style	Regular
Font Color	Black
Left	9.75
Top	0.2292

7 Select File | Save from the Delphi main menu.

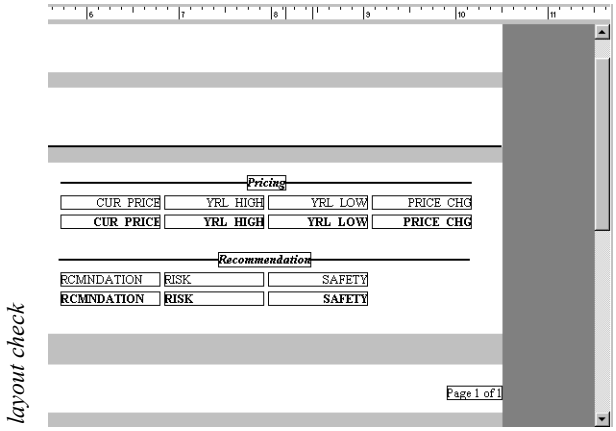
8 Preview the report by running the application.

PrintDateTime



layout check

PageSetDesc



layout check

Lay Out the Summary Band

1 Return to the Report Designer.

2 Select Report | Summary from the Report Designer main menu. A summary band should appear as the bottommost band in the Report Designer.

3 Right-click over the white space of the summary band and select the Position... menu option.

4 Set the height to 3.4063.

5 Scroll down so that you can see the entire summary band.

6 Place a label component on the left side of the summary band.

7 Configure the label component:

Caption	Summary Page
Font Name	Times New Roman
Font Size	16
Font Style	Bold
Font Color	Black
Text Alignment	Left justified
Left	0
Top	0.0313

8 Right-click over the white space of the summary band and select the NewPage menu option. This will cause the summary band to print on a new page at the end of the report.

9 Place a label component near the center of the summary band.

10 Configure the label component:

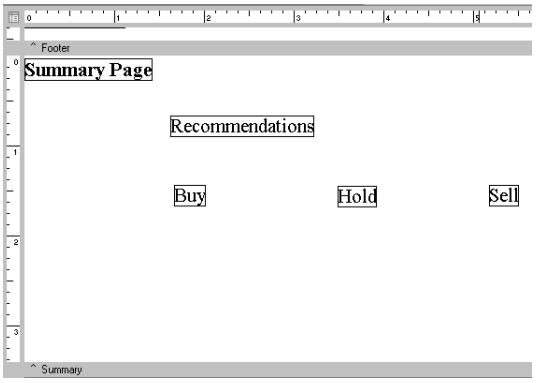
Caption	Recommendations
Font Name	Times New Roman
Font Size	16
Font Style	Regular
Font Color	Black
Text Alignment	Left justified
Left Position	1.625
Top	0.6667

11 Place three more label components in the summary band.

12 Set the captions:

Buy
Hold
Sell

layout check



Adjust the Summary Band Labels

1 Place the Buy label at the following position:

Left	2.8750
Top	1.0208

2 Place the Sell label at the following position:

Top	2.0104

3 Select the Buy, Hold, and Sell labels in that order.

4 Click the Align Right icon on the Align or Space toolbar.

5 Click the Space Vertically icon on the Align or Space toolbar.

6 Click the Right Justify icon on the Format toolbar.

7 Place a label component near the bottom of the summary band.

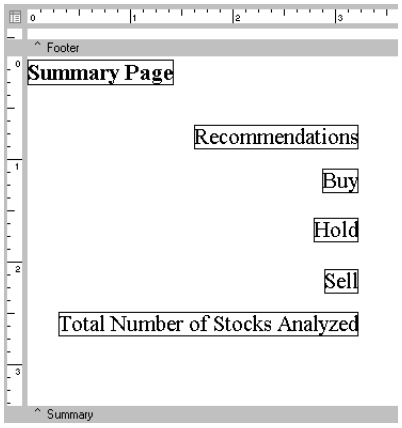
8 Configure the label component:

Caption	Total Number of Stocks Analyzed
Font Name	Times New Roman
Font Size	16
Font Style	Regular
Top Position	2.4896

9 Click on the Sell label, then Shift-click the new label.

10 Click the Align Right icon on the Align or Space toolbar.

layout check



Create and Adjust Variable Components

1 Place three Variable components to the right of the Buy, Hold, and Sell labels.

2 Set the names to vrBuyTotal, vrHoldTotal, and vrSellTotal in the Object Inspector.

3 Use the drop-down list in the Edit toolbar to set the DataType of each of the variables to Integer.

4 Select the vrBuyTotal variable and set the position:

Left	3.5521

5 Shift-click the vrHoldTotal and vrSellTotal variable components.

6 Click on the Align Right icon on the Align or Space toolbar.

7 Click the Right Justify icon on the Format toolbar.

8 Select the Buy label, then Shift-click the vrBuyTotal variable.

9 Click the Align Top icon of the Align or Space toolbar. Align the Hold and Sell variables with the corresponding labels.

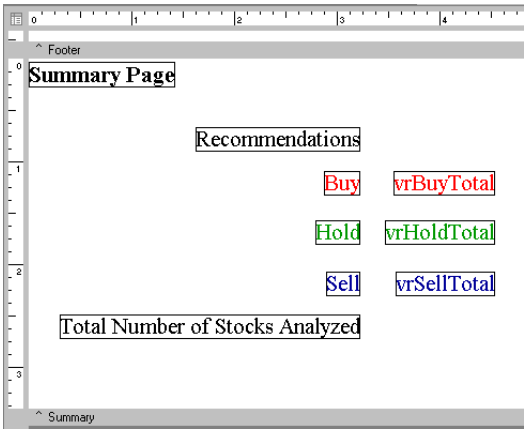
10 Select the Buy label and variable and set the font color to green.

11 Select the Hold label and variable and set the font color to olive.

12 Select the Sell label and variable and set the font color to maroon.

13 Select File | Save from the Delphi main menu.

layout check



Assign Event Handlers to the OnCalc Events of the Variable Components

1 Select the vrBuyTotal variable.

2 Click on the Events tab of the Object Inspector.

3 Double-click on the OnCalc event. An event handler shell will be generated in your Delphi form.

4 Assign the following code to this event handler:

```
if(qryStock.FieldByName('Rcmndation').AsString
  = 'BUY') then
   Value := Value + 1;
```

5 Assign the following code to the OnCalc event of the vrHold variable:

```
if(qryStock.FieldByName('Rcmndation').AsString
  = 'HOLD') then
   Value := Value + 1;
```

6 Assign the following code to the OnCalc event of the vrSell variable:

```
if(qryStock.FieldByName('Rcmndation').AsString
  = 'SELL') then
   Value := Value + 1;
```

Note: All three of these event handlers check the field value to make sure it corresponds to the total and then increment the total. The OnCalc event is a procedure where the Value parameter contains the current value of the variable. These particular variables are integers because we set the DataType to Integer when we created them.

7 Select File | Save from the Delphi main menu.

8 Select Project | Compile rbSTProj from the Delphi main menu. Fix any problems.

9 Run the project.

10 Preview the report. Click on the Last Page icon of the Print Preview window. A thirty-eight page report should appear. The summary band should contain totals for each recommendation type.

Note: You must compile and run this report in order to see exactly what it will look like. This is because event handlers have been assigned to the report. When event handlers are assigned, the report must be compiled and run; otherwise, the event handler code will not be used when generating the report and any changes made via the event handlers will not be reflected in the report.

Create the Grand Total

1 Return to the Report Designer.

2 Place a DBCalc component at the bottom of the summary band.

3 Right-click over the component and access the Calculations... menu option.

4 Select the Count function from the Calc Type drop-down list and click the OK button.

5 Configure the component:

Font Name	Times New Roman
Font Size	16
Font Style	Bold
Font Color	Black
Text Alignment	Right justified
Width	1.0729

6 Click on the vrSellTotal variable, then Shift-click the DBCalc component.

7 Click on the Align Right icon of the Align or Space toolbar.

8 Select the Total... label, then the DBCalc component.

9 Click on the Align Top icon of the Align or Space toolbar.

10 Place a line component at the bottom of the summary band.

11 Right-click over the line component and select the Double menu option. The line should now appear as two parallel lines.

12 Set the top of the line to 2.3437.

13 Select the DBCalc component, then Shift-click the line.

14 Select the View | Toolbars | Size menu option from the Report Designer main menu.

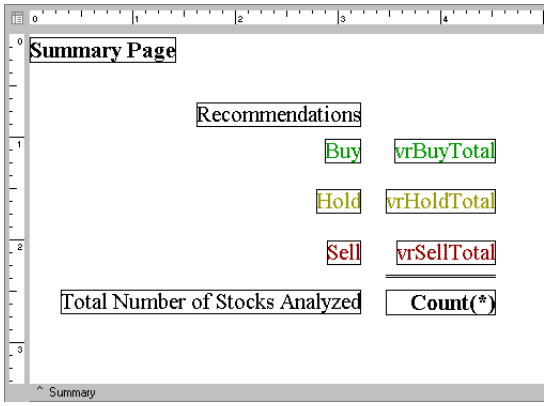
15 Dock the Size toolbar on the lower left side of the Report Designer.

16 Click on the Grow Width to Largest icon of the Size toolbar. The line should now be the same width as the DBCalc.

17 Click the Align Right icon on the Align or Space toolbar.

18 Select File | Save from the Delphi main menu.

layout check



Preview at Run-Time

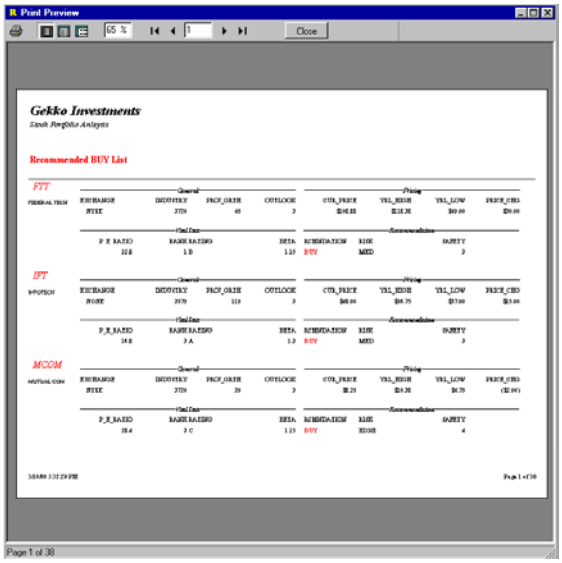
1 Select Project | Compile rbSTProj. Fix any compilation problems.

2 Select File | Save from the Delphi main menu.

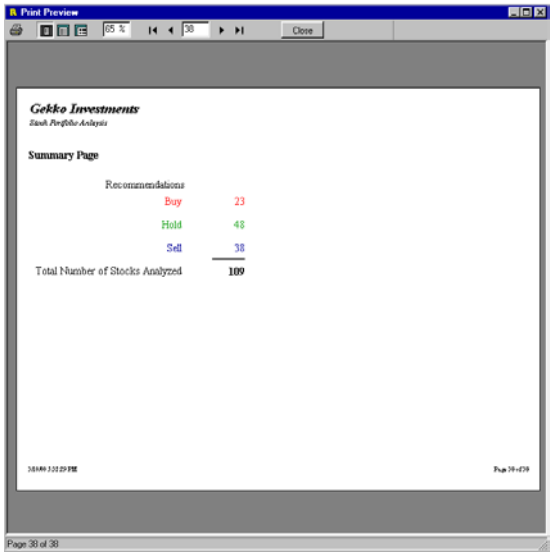
3 Run the project.

4 Preview the report. Click on the Last Page icon ⏭ of the Print Preview window. A grand total should appear at the bottom. The first page and last pages of the report should look like this:

First Page



Last Page

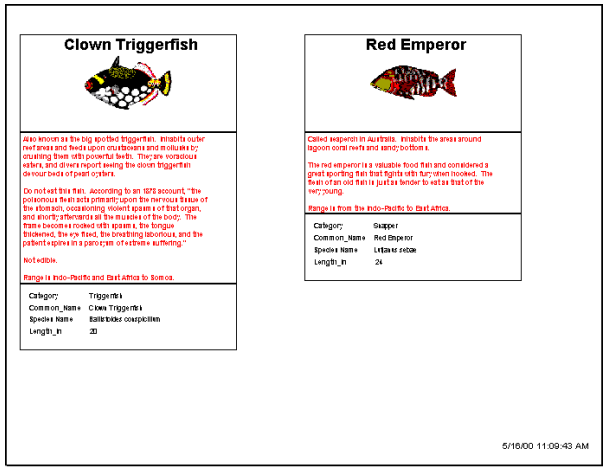


Using Regions to Logically Group Dynamic Components

Overview

This tutorial will show you how to do the following:

- Use regions
- Use a stretching memo
- Position components in relation to a stretching memo



Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmRegions'.

3 Select File | Save As from the Delphi menu and save the form under the name rbRegion in the My RB Tutorials directory (established on page 181).

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbRGProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create a Table, DataSource, and DataPipeline Component

1 Select the Data Access tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblBiolife
TableName	biolife.db

4 Select the Table component and then double-click on the Active property in the Object Inspector to set it to true.

5 Add a DataSource component to the form.

6 Configure the DataSource component:

DataSet	tblBiolife
Name	dsBiolife

7 Select the RBuilder tab of the Delphi component palette.

8 Add a DBPipeline component  to the form.

9 Configure the DBPipeline component:

DataSource	dsBiolife
Name	plBiolife

Create a Report and Connect it to the Data

1 Add a Report component  to the form.

2 Configure the Report component:

DataPipeline	plBiolife
Name	rbRegions

Invoke the Report Designer

1 Double-click on the Report component to display the Report Designer.

2 Size and move the Report Designer window so that the Object Inspector is visible.

3 Set the paper orientation to Landscape (File | Page Setup | Paper Size).

4 Click the Layout tab of the Page Setup dialog and set the Columns to 2.

5 Click the OK button to close the dialog.

Configure the Header and Footer Bands

1 Set the height of the header and footer bands to 0.3.

2 Scroll to the right in the Report Designer until you can see the right edge of the bands.

3 Place a SystemVariable component on the right side of the footer band.

4 Locate the drop-down list at the upper left corner of the Report Designer.

5 Select the PrintDateTime option from the drop-down list.

6 Configure the component:

Font Name	Arial
Font Size	12
Font Style	Regular
Font Color	Black
Left	8.9583
Top	0.0833
Text Alignment	Right justified

layout check

Create a Group on the 'Category' Field

1 Select Report | Groups from the main menu.

2 Select the 'plBiolife.Category' field from the drop-down list at the top of the dialog.

3 Click the Add button. A new group will be added to the list.

4 Click the Start New Column checkbox in the 'On Group Change' section of the dialog.

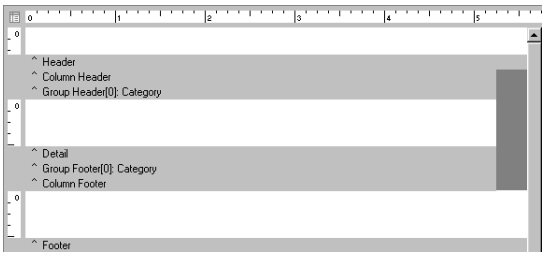
5 This option forces a new column each time the group changes, which allows each of the records from the Biolife table to appear in a separate column.

6 Click OK to close the Groups dialog.

7 Scroll to the left.

Note: You may notice that the group header and group footer bands created for the Category group are the same width as the column header, column footer, and detail bands. This is due to the fact that group bands print before and after the detail band. In order to print before and after the detail band, the group bands must be the same width as the detail band.

layout check



Create an Image Region

1 Set the height of the detail band to 3.5104.

2 Place a region component (from the Advanced toolbar) at the top of the detail band.

Note: A region is a special type of component that can contain other components. The region has the same printing behavior as a rectangular shape in that it can print a colored rectangle with a one pixel border. The gray shaded area you see around the edge of the region is not printed, but is provided so that you can distinguish a region from a shape component.

3 Configure the region:

Name	rgImage
Left	0
Top	0
Width	4
Height	1.8021

4 Select the View | Toolbars | Data Tree.

5 Position the Data Tree window to the left of the Report Designer (in front of the Object Inspector).

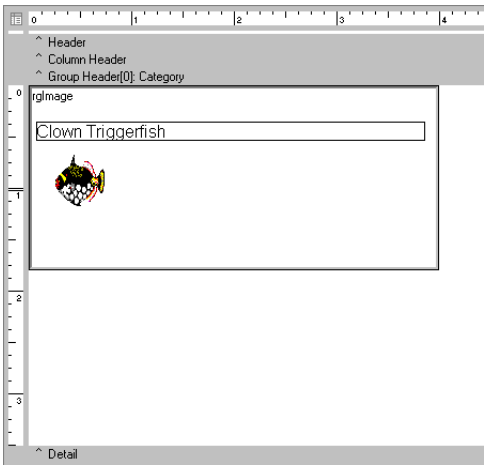
6 Click on the Layout tab of the Data Tree and click the Fields radio button.

7 Click on the Data tab.

8 Select the Common_Name and Graphic fields.

9 Drag the selection into the white space of the Region. If the Image component is created outside of the region, drag it into the region below the DBText component.

10 Click in the white space of the Region to deselect the components.



layout check

Adjust Image Region Components

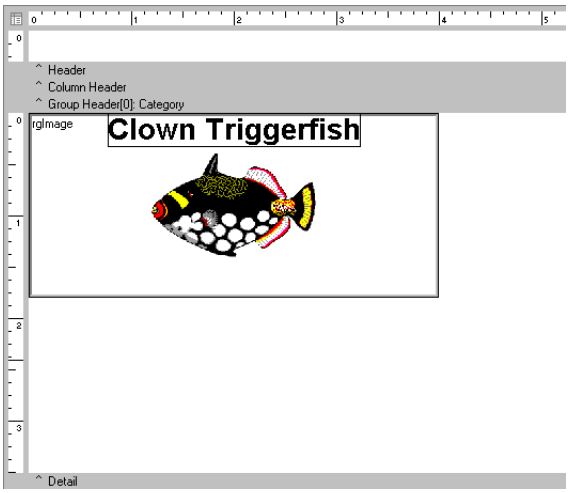
1 Configure the DBText component:

AutoSize	True
Font Name	Arial
Font Size	20
Font Style	Bold
Font Color	Black
Left	0.8229
Top	0.0208

2 Configure the DBImage component:

Left	1.125
Top	0.3854
Width	1.7604
Height	1.0313
MaintainAspectRatio	True

3 Select File | Save from the Delphi main menu.



layout check

Create a Memo Region

1 Place a region immediately below the image region.

2 Configure the region:

Name	rgMemo
Left	0
Top	1.8021
Width	4
Height	0.4687

Note: Notice that the region has a one pixel- wide black border. You can turn this border off by setting the Line Color to 'No Line' in the Report Designer or by setting the Pen.Style to psClear in the Object Inspector.

3 Select the Notes field in the Data Tree field list.

4 Drag the Notes field from the Data Tree into the white space of the memo region.

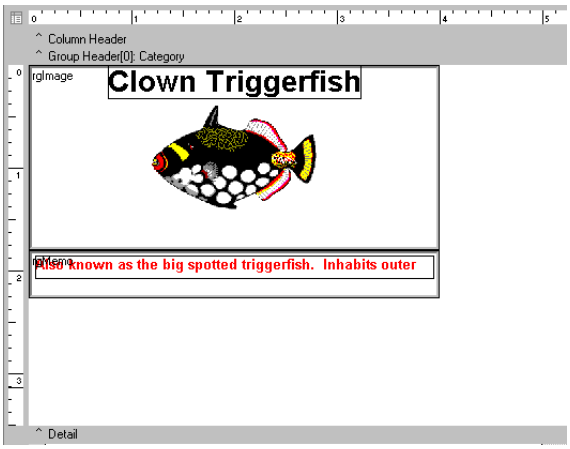
5 Configure the DBMemo component:

Name	dbmNotes
Stretch	True
Font Name	Arial
Font Size	10
Font Style	Bold
Font Color	Red
Left	0.0625
Top	1.8541
Width	3.8854
Height	0.2292
Bottom Offset	0.0208

6 Select File | Save from the Delphi main menu.

Note: By setting the Stretch property to True, we are telling the Memo to expand or contract based on the amount of text in the Notes field. The text will be wrapped within the width of the component. The Bottom Offset property determines the amount of white space that will appear below the memo after it has completed printing.

layout check



Create a Fields Region

1 Place a region in the space immediately below the memo region.

2 Configure the region:

Name	rgFields
Left	0
Top	2.2708
Width	4
Height	1.2396

3 Click on the Layout tab of the Data Tree.

4 Configure the Layout settings:

Create	All
Style	Vertical
Label Font Name	Arial
Label Font Style	Bold
Label Font Size	10
Label Font Color	Black
Field Font Name	Arial
Field Font Style	Regular
Field Font Size	10
Field Font Color	Black

5 Click on the Data tab to return to the field list.

6 Select the following fields and then drag the selection into the fields region:

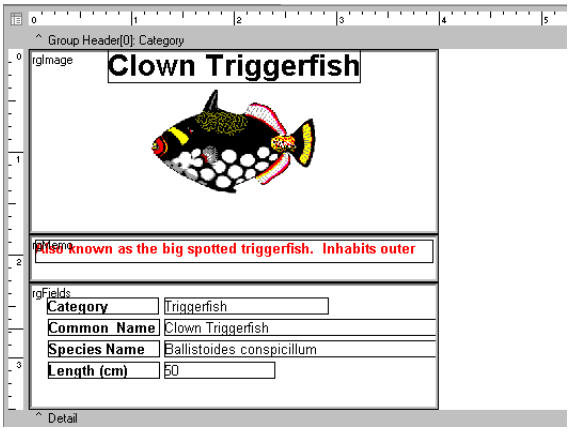
Category
Common_Name
Species Name
Length_In

7 Hold down the Ctrl key and use the arrow key to position the selection:

Left	0.1771
Top	2.4166

8 Close the Data Tree.

layout check



Adjust the DBText Components

1 Right-click over the 'Length_In' DBText component and set the DisplayFormat to #,0;-#,0 (the first DisplayFormat in the list).

2 Select all of the components in this region.

3 Set the AutoSize property to True in the Object Inspector.

4 Right-click over the white space of the Region and select the ShiftRelativeTo... menu option.

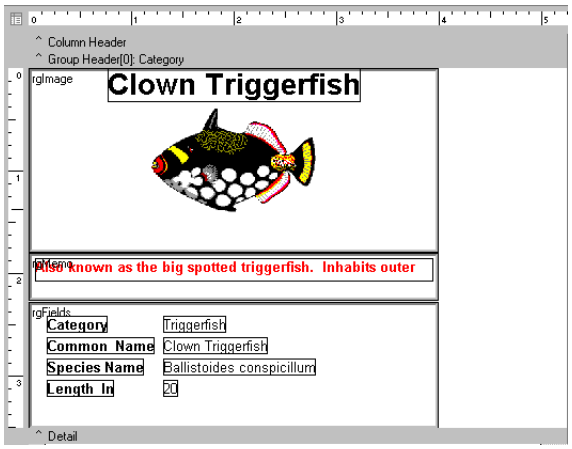
5 Select the rgMemo region from the drop-down list and then click the OK button.

Note: We set the Stretch property of the DBMemo component to True. That means that each time the memo prints, the height will expand or contract based on the amount of text in the Notes field. Because the memo is contained within a region, that region will also stretch to accommodate the memo. When that happens, we want to position the fields containing the DBText components immediately below the Memo region. We do this by setting the ShiftRelativeTo property of the Fields region to the Memo region.

6 Select File | Save from the Delphi main menu.

7 Close the Report Designer.

layout check



Add a Color-Coding Event Handler

1 Add the following variable to the private section of the form:

```
FColors: TList;
```

2 Add the following code to the OnCreate event of the form:

3 Add the following code to the OnDestroy event of the form:

```
procedure TfrmRegions.FormDe-
stroy(Sender:    ↳ TObject);
begin
   FColors.Free;
end;
```

Code | *Event handler for the OnCreate Event*

```
procedure TfrmRegions.FormCreate(Sender: TObject);
begin
   FColors := TList.Create;
   FColors.Add(TObject(clYellow));
   FColors.Add(TObject(clRed));
   FColors.Add(TObject(clLime));
   FColors.Add(TObject(clAqua));
   FColors.Add(TObject(clRed));
   FColors.Add(TObject(clSilver));
   FColors.Add(TObject(clYellow));
   FColors.Add(TObject(clRed));
   FColors.Add(TObject(clGray));
   FColors.Add(TObject(clGreen));
   FColors.Add(TObject(clOlive));
   FColors.Add(TObject(clGray));
   FColors.Add(TObject(clSilver));
   FColors.Add(TObject(clMaroon));
   FColors.Add(TObject(clSilver));
   FColors.Add(TObject(clAqua));
   FColors.Add(TObject(clRed));
   FColors.Add(TObject(clSilver));
   FColors.Add(TObject(clAqua));
   FColors.Add(TObject(clMaroon));
   FColors.Add(TObject(clGray));
   FColors.Add(TObject(clBlue));
   FColors.Add(TObject(clYellow));
   FColors.Add(TObject(clRed));
   FColors.Add(TObject(clSilver));
   FColors.Add(TObject(clMaroon));
   FColors.Add(TObject(clYellow));
   FColors.Add(TObject(clSilver));
end;
```

4 Double-click on the Report component to access the Report Designer.

5 Click in the white space of the detail band.

6 Select the Events tab of the Object Inspector and double-click on the BeforeGenerate event. Add the following code:

Code | *Event handler for the BeforeGenerate Event*

```
procedure TfrmRegions.ppDetailBand1BeforeGenerate(Sender: TObject);
var
   lColor: TColor;
begin

   if not(rbRegions.DetailBand.OverFlow) then
     begin
       lColor := TColor(FColors[plBiolife.TraversalCount]);

       rgFields.Brush.Color := lColor;
       dbmNotes.Font.Color := lColor;
     end;
end;
```

Note: This event handler sets the color of the Fields region and the Memo font. The colors were hand-picked to match the bitmap images. Because there is only one record per column, the DataPipeline.TraversalCount property (which tracks the number of records traversed) can be used to map the color from the TList object to the column.

Preview the Report at Run-Time

1 Select the Standard tab of the Delphi component palette.

2 Add a Button component to the form.

3 Configure the Button component:

Name	btnPreview
Caption	Preview

4 Add the following code to the OnClick event handler of the button:

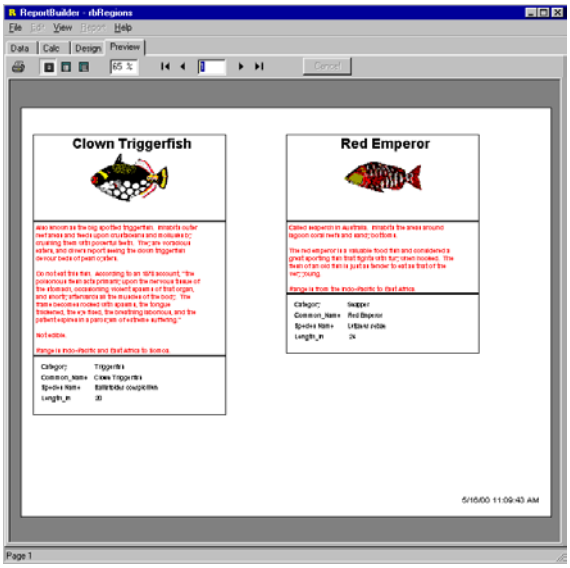
```
rbRegions.Print;
```

5 Select Project | Compile rbRGProj. Fix any compilation problems.

6 Select File | Save from the Delphi main menu

7 Run the project..

8 Click on the Preview button. Click the Last Page icon  . The report should be fourteen pages, with two fish per page. The Memo font and the bottommost region should be color-coded. The report should look like this:

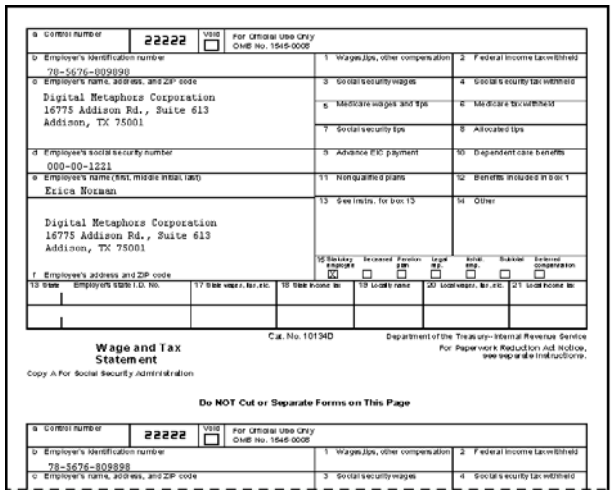


Forms Emulation with a WMF Image

Overview

This tutorial will show you how to do the following:

- Emulate forms using Windows metafiles
- Perform calculations at the dataset level
- Dynamically format an address using a memo



Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmFormsEmulation'.

3 Select File | Save As from the Delphi menu and save the form under the name rbFormE in the My RB Tutorials directory (established on page 181).

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbFEProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create a Table, DataSource, and DataPipeline Component

1 Select the Data Access tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblCustomer
TableName	customer.db

4 Add a DataSource component to the form.

5 Configure the DataSource component:

DataSet	tblCustomer
Name	dsCustomer

6 Select the RBuilder tab of the Delphi component palette.

7 Add a DBPipeline component  to the form.

8 Configure the DBPipeline component:

DataSource	dsCustomer
Name	plCustomer

Create a Report and Connect it to the Data

1 Add a Report component  to the form.

2 Configure the Report component:

DataPipeline	plCustomer
Name	rbFormEmu

Create Calculated Fields

1 Double-click on the table component. The Field Editor will be displayed.

2 Right-click over the white space of the Field Editor and select the Add Fields... menu option. Add all of the fields listed.

3 Right-click over the white space of the Field Editor and select the New Field... menu option.

4 Type Wages in the Name Edit Box. Set the DataType to Currency. The Field Type should default to Calculated.

5 Click OK.

6 Repeat this process to create the following fields:

SSWages
MedicareWages
FederalTaxWithheld
SSTaxWithheld
MedicareTaxWithheld

7 Close the Field Editor. Select the table component.

8 Click on the Events tab of the Object Inspector.

9 Double-click in the OnCalcFields event. An event handler will be generated. Add the code as shown below.

Code — *Event handler for the OnCalcFields Event*

```
procedure TfrmFormsEmulation.tblCustomerCalcFields(DataSet: TDataSet);
begin
   tblCustomerWages.AsCurrency := 40000;
   tblCustomerSSWages.AsCurrency := 40000;
   tblCustomerMedicareWages.AsCurrency := 40000;

   tblCustomerFederalTaxWithheld.AsCurrency := 40000 * 0.2;
   tblCustomerSSTaxWithheld.AsCurrency := 40000 * 0.05;
   tblCustomerMedicareTaxWithheld.AsCurrency := 40000 * 0.01;
end;
```

Note: This event handler executes every time the record position of the customer table changes. Calculations placed in this event must always be intra-record, which means that the values must be calculated from other values in the current record or must be hard-coded (as in this case).

10 Double-click the DBPipeline component to display the Field Editor.

11 Check to make sure that the calculated fields are listed. If they are not, then close the Field Editor; set the AutoCreateFields property to False in the Object Inspector, then set it back to True. Launch the Field Editor again and make sure the new fields are listed.

Note: Toggling the AutoCreateFields property for the DBPipeline will refresh the field list whenever you've changed the fields in the dataset.

12 Select Project | Compile rbFEProj from the Delphi main menu. Fix any compilation errors.

13 Select File | Save from the Delphi main menu.

Configure the Page Size and Bands

1 Double-click on the Report component to display the Report Designer.

2 Size and move the Report Designer window so that the Object Inspector is visible.

3 Select Report | Header from the Report Designer main menu. The header band will be removed.

4 Select Report | Footer from the Report Designer main menu. The footer band will be removed.

5 Click the report selection icon located at the intersection of the horizontal and vertical rulers. The report will be selected in the Object Inspector.

6 Expand the PrinterSetup property and set all of the margins (MarginTop, MarginBottom, etc.) to zero.

7 Set the height of the detail band to 11.

Create an Image Component

1 Place an image component on the left side of the detail band.

2 Configure the Image component:

Center	False
DirectDraw	True
MaintainAspectRatio	True
Stretch	True

Note: We want the image to be scaled because it's a little too big to fit on the page, despite the zero margins. We do not want to distort the form in any way, so MaintainAspectRatio has been set to True. Center is not needed because we want to print the form at a position of 0,0 (this form image already contains white space around the edges). Finally, the DirectDraw property will force the image to be printed directly to the printer without utilizing an intermediate canvas, which usually results in higher quality output.

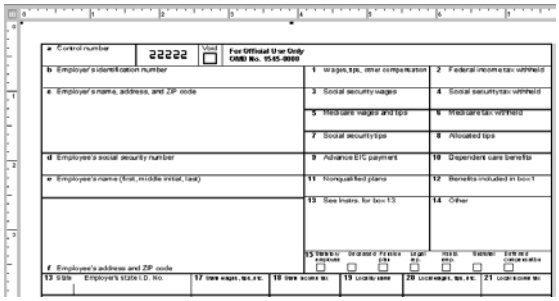
3 Right-click over the Image component and select the Picture... menu option.

4 Open the W2.WMF file. This file should be located in the RBuilder\Tutorials directory.

5 Set the image component position:

Left	0
Top	0
Width	7.8125
Height	10.9792

layout check



Create the Wages DBText Components

1 Select View | Toolbars | Data Tree from the Report Designer main menu.

2 Position the Data Tree to the left of the Report Designer (in front of the Object Inspector).

3 Click on the Layout tab of the Data Tree and configure it:

Create	Fields
Style	Vertical
Field Font Name	Courier New
Field Font Style	Regular
Field Font Size	10
Field Font Color	Black

4 Click the Data tab. Select the following fields in order:

Wages
SSWages
MedicareWages

5 Drag the fields into the detail band, positioning the mouse over the area of the form labeled '1 Wages, tips and other compensation'. Release the mouse button.

6 Close the Data Tree.

7 Use the Object Inspector to set the width to 1.5208. The width of the selected components will increase.

8 Right-click over the Wages DBText component and select the DisplayFormat menu option.

9 Set the Display Format to $#,0.00;($#,0.00), (the first selection with a dollar sign) select the text in the Edit box, and copy it to the clipboard. Click Cancel.

10 Click on the DisplayFormat property in the Object Inspector. The selected DisplayFormat should appear. Paste the DisplayFormat into this field and then click off of the property.

11 Click on the Image and then click on each of the DBText components in succession, checking to make sure that the DisplayFormat has been set.

12 Set the position of the Wages DBText component:

Left	4.375
Top	0.8021

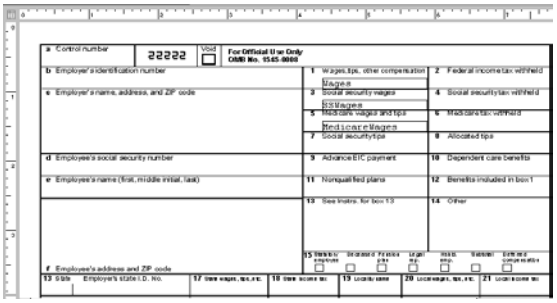
13 Set the position of the SSWages DBText component:

Left	4.375
Top	1.1146

14 Set the position of the MedicareWages DBText component:

Left	4.375
Top	1.4271

layout check



Create the Withholding DBText Components

1 Select the Wages, SSWages, and MedicareWages DBText components.

2 Copy and paste the selection.

3 Drag the selection to the right, positioning it over '2 Federal income tax withheld' area of the image.

4 Set the width of the components to 1.6042 in the Object Inspector.

5 Hold down the Ctrl key and use the arrow keys to position the components:

Left	6.0521
Top	0.8021

6 Set AutoSize to True in the Object Inspector. All three DBText components should be autosized.

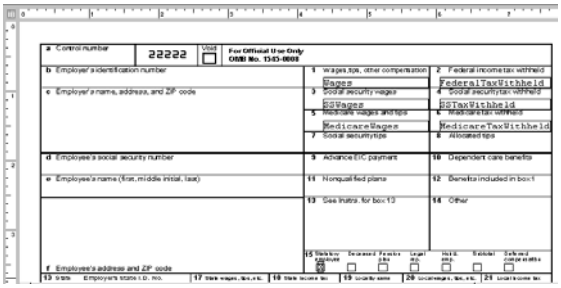
7 Click on the image to deselect the components.

8 Select the newly created Wages DBText component and set the field to FederalTaxWithheld.

9 Select the SSWages DBText component and set the field to SSTaxWithheld.

10 Select the MedicareWages DBText component and set the field to MedicareTaxWithheld.

layout check



Create the Address Information

1 Scroll to the top and left so that the entire left corner of the image is visible.

2 Place a label in 'b Employee's identification number' box.

3 Configure the label:

Caption	78-5676-809898
Left	0.5729
Top	0.8021

4 Place a memo component in the Employer's address box of the form (box c).

5 Set the memo position:

Left	0.5313
Top	1.125
Width	3.5521
Height	0.7708

6 Right-click over the memo and access the Lines... menu option. Enter the following text into the Memo Editor:

Digital Metaphors Corporation
16775 Addison Rd., Suite 613
Addison, TX 75001

7 Select the memo component.

8 Copy and paste the selection.

9 Drag the new memo component to the Employee's address box (box f).

10 Set the memo position:

Left	0.5
Top	2.7813

11 Set the memo name to mmEmployeeAddress1.

12 Place a DBText component in the Employee's social security number box (box d).

13 Configure the DBText component:

AutoSize	True
DataField	CustNo
DisplayFormat	000-00-0000
Left	0.5
Top	2.0416

14 Place a DBText component in the Employee's name box (box e).

15 Configure the DBText component:

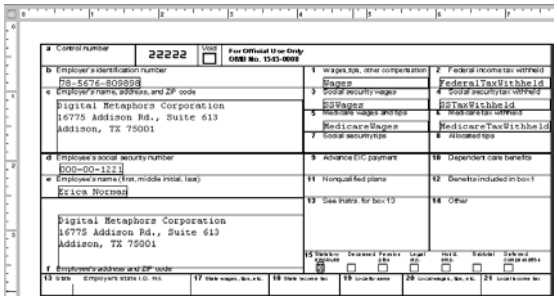
AutoSize	True
DataField	Contact
Left	0.5
Top	2.3646

16 Place a label in box '15' of the form. Set the Caption to X.

17 Position the label:

Left	4.2917
Top	3.4583

layout check



Create Duplicate Component Information

1 Select all of the components on the form image.

2 Copy and paste the selection.

3 Scroll down to the duplicate form.

4 Position the selection within the boxes of the duplicate form.

5 Hold down the arrow keys and position the selection:

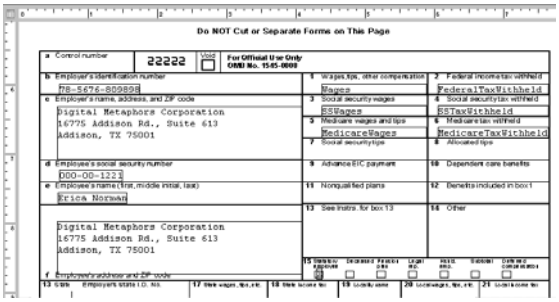
Left	0.5
Top	5.9896

6 Select the memo in the Employee's Address box of the duplicate form (box f).

7 Set the memo name to mmEmployeeAddress2.

8 Select File | Save from the Delphi main menu.

layout check



Write the 'address squeeze' Routine

1 Close the Report Designer.

2 Locate the drop-down list of components at the top of Object Inspector.

3 Select the Detail band in this list (it should be named ppDetailBand1).

4 Select the Events tab and double click on the BeforeGenerate event.

5 Place the following code in the event handler:

```
BuildEmployeeAddress(mmEmployeeAddress1.Lines);
BuildEmployeeAddress(mmEmployeeAddress2.Lines);
```

6 Scroll up to the form class declaration. Replace the private declarations comment with the following procedure declaration:

```
procedure BuildEmployeeAddress(aStrings:
↳  TStrings);
```

7 Scroll down below the DetailBand BeforeGenerate event handler and insert the code shown below as the BuildEmployeeAddress procedure.

Note: In this event handler we need to build the same address for two different memo components. In order to accomplish this, we can create a

Code | *Event handler for the BuildEmployeeAddress Procedure*

```
procedure TfrmFormsEmulation.BuildEmployeeAddress(aStrings: TStrings);
var
   lsLine: String;
   lsState: String;
   lsZIP: String;

begin
   {clear memo}
   aStrings.Clear;

   {add contact}
   lsLine := tblCustomer.FieldByName('Contact').AsString;
   aStrings.Add(lsLine);

   {add address line1}
   lsLine := tblCustomer.FieldByName('Addr1').AsString;
   if (lsLine <> '') then
     aStrings.Add(lsLine);

   {add address line2}
   lsLine := tblCustomer.FieldByName('Addr2').AsString;
   if (lsLine <> '') then
     aStrings.Add(lsLine);

   {add city, state zip}
   lsLine := tblCustomer.FieldByName('City').AsString;

   lsState := tblCustomer.FieldByName('State').AsString;

   if (lsState <> '') then
     lsLine := lsLine + ', ' + lsState;

   lsZIP := tblCustomer.FieldByName('ZIP').AsString;

   if lsZIP <> '' then
     lsLine := lsLine + '' + lsZIP;

   aStrings.Add(lsLine);

end;
```

general routine (BuildEmployeeAddress) that we can call for each component, or we can build the address in a local string variable and then assign it to both components. We chose the latter approach for readability.

Note: This routine simply retrieves each element of the Employee's address, concatenating and storing the result in the TStrings object passed in the parameter. The 'if' statements check for empty strings, ensuring that no blank lines will appear in the address.

Preview the Report at Run-Time

1 Select the Standard tab of the Delphi component palette.

2 Add a Button component to the form.

3 Configure the Button component:

Name	btnPreview
Caption	Preview

4 Add the following code to the OnClick event handler of the button:

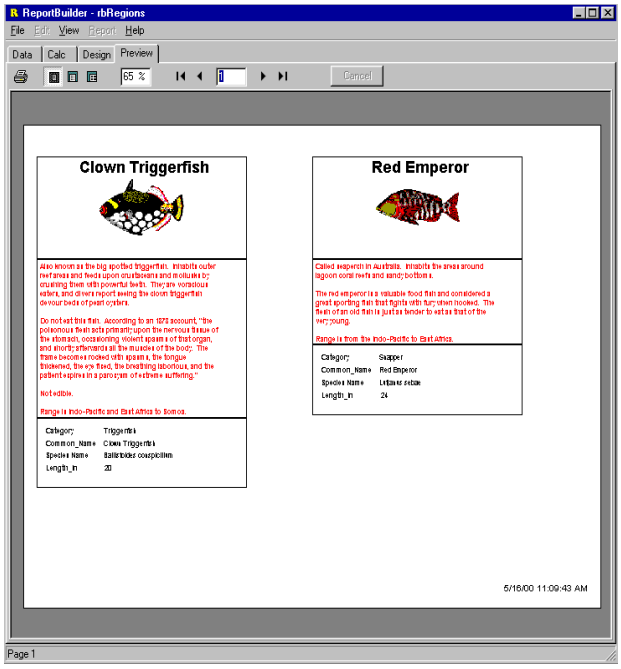
```
rbFormEmul.Print;
```

5 Select Project | Compile rbFEProj. Fix any compilation problems.

6 Select File | Save from the Delphi main menu.

7 Run the project.

8 Click on the Preview button. The report should be displayed in the Print Preview form. The report should look like this:

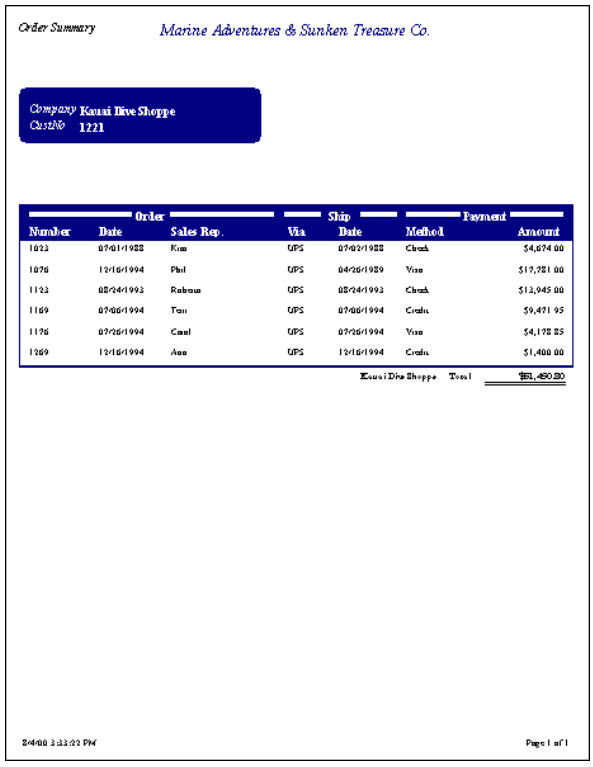


Master → Detail Report

Overview

This tutorial will show you how to do the following:

- Configure Delphi data access objects for use in a master/detail report
- Configure Delphi data access objects for use as a lookup table
- Use a subreport to print detail data
- Define a group break



Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmMasterDetail'.

3 Select File | Save As from the Delphi menu and save the form under the name rbMD in the My RB Tutorials directory (established on page 181).

Note: It is important to name the form and save the form's unit using the names given above because the report will be used in later tutorials.

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbMDProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create the Table, DataSource, and DataPipeline for the Master Table

1 Select the Data Access tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblCustomer
TableName	customer.db

4 Add a DataSource component to the form.

5 Configure the DataSource component:

DataSet	tblCustomer
Name	dsCustomer

6 Select the RBuilder tab of the Delphi component palette.

7 Add a DBPipeline  component to the form.

8 Configure the DBPipeline component:

DataSource	dsCustomer
Name	plCustomer

Create the Table, DataSource, and DataPipeline for the Detail Table

1 Add a second set of Table, DataSource, and DataPipeline components to your form.

2 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblOrder
TableName	orders.db

3 Configure the DataSource component:

DataSet	tblOrder
Name	dsOrder

4 Configure the DBPipeline component:

DataSource	dsOrder
Name	plOrder

Create the Table, DataSource, and DataPipeline for the Lookup Table

1 Add a third set of Table, DataSource, and DataPipeline components to your form.

2 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblEmployee
TableName	employee.db

3 Configure the DataSource component:

DataSet	tblEmployee
Name	dsEmployee

4 Configure the DBPipeline component:

DataSource	dsEmployee
Name	plEmployee

Define the Relationship between the Customer and Order Table

1 Select the Order table component.

2 Set the MasterSource property to dsCustomer.

3 Select the MasterFields property and click the "..." button. This displays the Field Link Designer.

4 Click on the Available Indexes drop-down list and select CustNo.

5 Select the CustNo in the Detail Fields list.

6 Select the CustNo item in the Master Fields list.

7 Click the Add button. A new item (CustNo→ CustNo) will be added to the Joined Fields list.

8 Click the OK button to close the dialog.

Note: The IndexName property is now set to CustNo. This property was automatically set when we selected the CustNo index in step 4 above.

Note: To establish a master detail relationship, the detail table must be indexed on the linking field value or values. You can think of an Index as defining the sort order for the table. Thus, if we need to link the Order table to the Customer table based on CustNo, the Order table must have an index (i.e. sort order) based upon CustNo.

Define the Relationship between the Order and Employee Table

1 Select the Employee table component.

2 Set the MasterSource property to dsOrder.

3 Launch the Field Link Designer from the MasterField property.

4 Select the EmpNo item field in the Detail Fields list.

5 Select the EmpNo field in the Master Fields list.

6 Click the Add button. A new item (EmpNo→ EmpNo) will be added to the Joined Fields list.

7 Click the OK button to close the dialog.

8 Select File | Save from the Delphi main menu.

Note: In this example you will notice the fields being linked have the same name. This is a good standard to follow when designing a database. However, it is not a requirement for linking fields.

Create a Report and Connect it to the Data

1 Add a Report component  to the form.

2 Configure the Report component:

DataPipeline	plCustomer
Name	rbOrderSummary

3 Double-click on the Report component to display the Report Designer.

4 Position the Report Designer so that the Object Inspector is visible.

5 Place a subreport  component (from the Advanced toolbar) in the detail band.

Note: When you add the subreport, two tabs will be displayed at the bottom of the workspace: Main and SubReport1. There is a separate workspace for the main report and for each subreport. You can access the workspace for any given report by clicking on the tab.

6 Use the Object Inspector to set the DataPipeline property of the subreport to plOrder.

Note: When we initially set the DataPipeline property of the main report to the customer table, we were telling the report to print one detail band for each customer. Now that we have set the DataPipeline property of the subreport, we're telling the subreport to traverse all orders for each customer, printing one detail band for each order.

7 Select Report | Data from the main menu. plCustomer should be selected as the data pipeline.

Note: The Data dialog can be used as a convenient way to set the DataPipeline property of a report.

8 Position the subreport at the top of the detail band.

9 Drag the the detail band divider up until it is flush with the bottom of the subreport.

layout check

Create the Header Band Labels

1 Set the height of the header band to 2.5 inches.

2 Place a label in the upper left corner of the header band.

3 Configure the label:

Caption	Order Summary
Font Name	Times New Roman
Font Size	12
Font Style	Bold & Italic

4 Place another label at the top center of the header band.

5 Configure the label:

Caption	Marine Adventures & Sunken Treasure Co.
Font Name	Times New Roman
Font Size	16
Font Style	Bold & Italic
Font Color	Navy
Text Alignment	Centered

6 Center the label by clicking the Center Horizontally in Band icon on the Align or Space toolbar.

7 Select both labels and drag them to the top of the header band.

layout check

Create the Header Band Shape

1 Place a shape component in the left side of the header band.

2 Select 'Rounded Rectangle' from the drop-down list at the upper left corner of the Report Designer.

3 Use the Draw toolbar to set the Fill and Line color of the shape to Navy.

4 Drag the shape to the left edge of the band and position the top at the 1 inch mark on the vertical ruler.

5 Set the size of the shape:

Width	3.5
Height	0.8

6 Select File | Save from the Delphi main menu.

layout check

Use the Data Tree to Complete the Header Band Layout

1 Select View | Toolbars | Data Tree from the Report Designer menu. Position the Data Tree to the left of the Report Designer, in front of the Object Inspector.

2 Click the Layout tab at the bottom of the Data Tree and configure the drag-and-drop settings:

Create	All
Style	Vertical
Label Font Name	Times New
Label Font Style	Italic
Label Font Size	12 point
Label Font Color	White
Field Font Name	Times New
Field Font Style	Bold
Field Font Size	12 point
Field Font Color	White

3 Click on the Data tab of the Data Tree. Make sure that the Customer data pipeline is selected.

4 Click on the Company field and then Ctrl-click the CustNo field.

5 Drag the selected fields onto the workspace, positioning the mouse just inside the upper left corner of the shape. Release the mouse button. Two DBText components and corresponding labels will be created.

6 Hold down the Ctrl key and use the arrow keys to position the selection:

Left	0.1458
Top	1.1875

7 Click in the white space of the header band to deselect the components.

8 Select the CustNo label, then Shift-click the corresponding DBText component. Click on the Left Justify icon on the Format toolbar.

9 Right-click over the Company DBText component and set the AutoSize property to True.

10 Right-click over the CustNo DBText component and set the AutoSize property to True.

layout check

Create a Group on the CustNo Field

1 Select Report | Groups from the Report Designer main menu. The Groups dialog will be displayed.

2 Select the 'plCustomer.CustNo' field from the drop-down list at the top of the dialog.

3 Click the Add button. A new group will be created.

4 Click the Start New Page and Reset Page Number check boxes near the middle of the dialog. This will cause the listing for each customer to start on a new page and appear with its own page numbers.

Note: Because groups are based on a change in field value, the field on which a group is based should always be used to order the records. ReportBuilder does not sort records; therefore, you must order the records using an Index or by specifying an Order By clause in an SQL query.

5 Click the OK button. You should see two new bands: one above and one below the detail band. These bands have no height; thus, no white space appears above the dividers.

6 Select File | Save from the Delphi main menu.

layout check

Begin the Order Information Layout

1 Click the 'SubReport1' tab at the bottom of the designer.

Note: The selected object in the Object Inspector will change to ppChildReport1. The subreport component is a container for the child report. The subreport controls the properties that direct the print behavior of the report. The subreport also controls the report's relationship to other components within the main report layout. The child report contains the actual report layout (bands, labels, etc.) and is accessible via the Report property of the subreport component (i.e. SubReport1.Report).

2 Place a shape component in the left side of the title band.

3 Use the Draw toolbar to set the Fill and Line color of the shape to Navy.

4 Right click over the shape to display the popup menu. Select the ParentHeight and ParentWidth options. The shape will resize and cover the entire band.

5 Click the Layout tab at the bottom of the Data Tree and configure the drag-and-drop settings:

Create	All
Style	Tabular
Label Font Name	Times New
Label Font Style	Bold
Label Font Size	12 point
Label Font Color	White
Field Font Name	Times New
Field Font Style	Regular
Field Font Size	10 point

6 Click the Data tab at the bottom of the Data Tree and select the plOrder data pipeline.

7 Use the Ctrl-click method to select these fields:

OrderNo
SaleDate

8 Drag the selected fields into the title band, positioning the mouse at the bottom left corner of the shape. Release the mouse button to create the components.

9 Hold down the Ctrl key and use the arrow keys to position the selection:

Left	0.1458
Top	0.2813

10 Set the label captions to Number and Date.

11 Left justify all of the components.

12 Right-click over the SaleDate DBText component and set the DisplayFormat to 'mm/dd/yyyy'.

Note: The Display Format dialog automatically checks the data type of the component and displays a list of appropriate formats.

13 Set the width of the Date label and the corresponding DBText component to 0.8.

14 Select the DBText components and move them to a top position of 0.0521.

layout check

Complete the Order Information Layout

1 Select the plEmployee DataPipeline in the Data Tree.

2 Drag the FirstName field into the title band, positioning the mouse to the immediate right of the Date label.

3 Position the selection:

Left 2.2083
Top0.2813

4 Align the top of the new DBText with the top of the Date DBText component.

5 Set the label Caption to Sales Rep.

6 Place a shape component in the left side of the title band.

7 Align the left edge of the shape with the Number label and stretch the width until the right edge is flush with the right edge of the Sales Rep label.

8 Configure the shape:

Top	0.1
Height	0.08
Fill Color	White
Line Color	White

9 Place a label directly on top of the shape, positioning it near the shape's midpoint.

10 Configure the label:

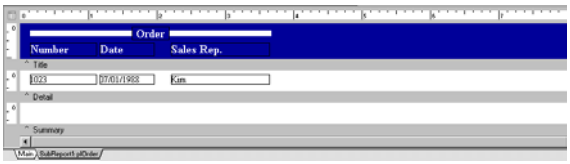
AutoSize	False
Caption	Order
Font Name	Times New Roman
Font Size	12
Font Style	Bold
Font Color	White
Highlight Color	Navy
Text Alignment	Centered
Width	0.55

11 Select the white shape, then Shift-click the Order Label. Click on the Align Middle and Align Center icons of the Align or Space toolbar.

12 Select File | Save from the Delphi main menu.

13 Click the Preview tab to view the report. Click the next page buttons to preview several pages. When you are done previewing, return to the Design workspace.

layout check



Lay Out the Shipping Information Components

1 Select the plOrder DataPipeline in the Data Tree and select the following fields in order:

ShipVia
ShipDate

2 Drag the selection into the title band, to the immediate right of the Sales Rep label. Position the selection:

Left	3.875
Top	0.2813

3 Align the top of the new DBText components with the top of the existing components.

4 Set the label captions to Via and Date.

5 Set the width of the Date label and corresponding DBText component to 0.8.

6 Set the Display Format of the Date DBText component to 'mm\dd\yyyy'.

7 Select the Order label, then Shift-click the white shape behind it.

8 Copy and paste the selection.

9 Position the new components above the Via and Date labels.

10 Set the position and size of the shape:

Left	3.8333
Width	1.61

11 Align the top of the new shape with the top of the existing shape.

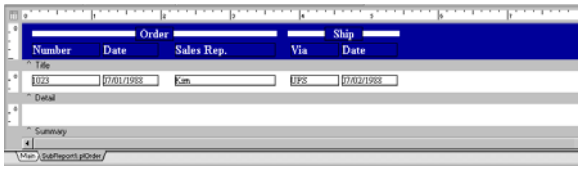
12 Right-click over the label and select the Bring to Front menu option.

13 Set the label caption to Ship.

14 Select the shape component, then Shift-click the Ship label. Click on the Align Middle and Align Center icons of the Align or Space toolbar.

15 Select File | Save from the Delphi main menu. Preview as desired.

layout check



Lay Out the Payment Information Components

1 Select the plOrder DataPipeline in the Data Tree and select the following fields in order:

PaymentMethod
AmountPaid

2 Drag the selection into the title band, to the immediate right of the ShipDate label. Position the selection:

Left	5.5833
Top	0.2813

3 Align the top of the new DBText components with the top of the existing components.

4 Set the label captions to Method and Amount.

5 Set the DisplayFormat of the Amount DBText component to $#,0.00;($#,0.00) (the first format with a dollar sign).

6 Right justify the text in the Amount label.

7 Select the Ship label, then Shift-click the white shape behind it.

8 Copy and paste the selection.

9 Drag the new components to the right, aligning the left edge of the shape with the left edge of the Method label.

10 Align the top of the new shape with the top of the existing shapes.

11 Set the width of the new shape to 2.27.

12 Right-click over the label and select Bring to Front.

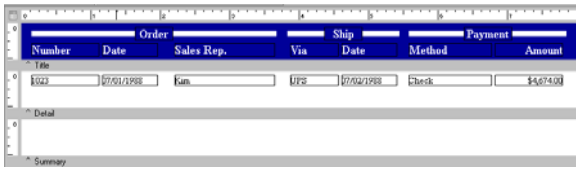
13 Configure the label:

Caption	Payment
Width	0.7

14 Select the shape, then Shift-click the label. Click on the Align Middle and Align Center icons of the Align or Space toolbar.

15 Select File | Save from the Delphi main menu. Preview as desired.

layout check



Complete the Detail Band

1 Place a Line component in the upper left corner of the detail band.

2 Locate the Line Position drop-down list at the upper left of the Report Designer.

3 Select Left from this list. A vertical line will appear on left side of the line component.

4 Configure the Line:

Color	Navy
Thickness	1 1/2 point
Left	0
Width	0.10
ParentHeight	True

5 Copy and paste the line component.

6 Select Right from the Line Position drop-down list. The vertical line will move to the right side of the line component.

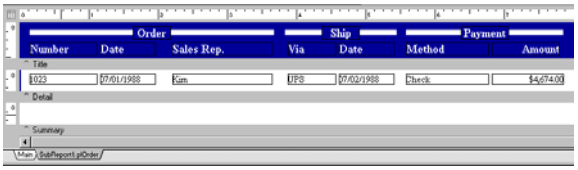
7 Drag the new line to the right side of the detail band. Hold down the Ctrl key and use the right arrow button to position the line so it is flush with the right edge of the band. Preview to make sure the shape and the line in the title band are flush.

8 Set the height of the detail band to 0.3.

9 Set the height of the summary band to 0.3125.

10 Select File | Save from the Delphi main menu. Preview as desired.

layout check



Lay Out the Summary Band

1 Place a DBCalc component in the summary band.

2 Configure the DBCalc component:

DataField	Amount Paid
Font Name	Times New Roman
Font Size	10
Font Style	Bold
Font Color	Black
Top	0.083
AutoSize	True
DisplayFormat	$#,0.00;($#,0.00)
Text Alignment	Right justified

3 Select the AmountPaid DBText component in the detail band, then Shift-click the DBCalc component. Click on the Grow Width to Largest icon of the Size toolbar. Click on the Align Right icon of the Align or Space toolbar.

4 Place a line directly below the DBCalc component.

5 Configure the Line:

Position	Bottom
Style	Double
Top	0.25
Width	1.18
Height	0.05

6 Hold down the Ctrl key and use the right arrow key to align the right edge of the line with the right edge of the DBCalc.

7 Place a label to the left of the DBCalc component. Configure the label:

Caption	Total
Font Name	Times New Roman
Font Size	10
Font Style	Bold
Left	6.2292

8 Align the top of the label with the top of the DBCalc.

9 Place a DBText component to the left of the label. Configure the DBText:

DataPipeline	plCustomer
DataField	Company
Font Name	Times New Roman
Font Size	10
Font Style	Bold
Left	5
Autosize	True
Text Alignment	Right justified

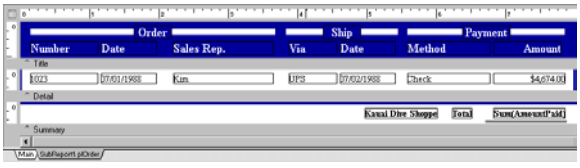
10 Align the top of the DBText component with the top of the label.

11 Place a line component in the upper left corner of the summary band. Configure the line:

ParentWidth	True
Line Color	Navy
Line Thickness	2 1/4 point
Top	0
Height	0.05

12 Select File | Save from the Delphi main menu. Preview as desired.

layout check



Lay Out the Footer Band

1 Click on the 'Main' tab to return to the main report.

2 Place a SystemVariable component in the lower left corner of the footer band.

3 Select PrintDateTime from the drop-down list at the upper left corner of the Report Designer.

4 Set the left position to 0.0625.

5 Set the font:

Font Name	Times New Roman
Font Size	10
Font Color	Black
Font Style	Regular
Text Alignment	Left justified

6 Place another System Variable component in the lower right corner of the footer band.

7 Select PageSetDesc from the drop-down list at the upper left corner of the Report Designer.

8 Set the left position to 7.3229.

9 Set the font:

Font Name	Times New Roman
Font Size	10
Font Color	Black
Font Style	Regular
Text Alignment	Right justified

10 Align the tops of the System Variable components.

11 Drag the footer band divider up until it meets the bottom of the System Variable components.

12 Select File | Save from the Delphi main menu. Preview the completed report.

layout check

Preview the Report at Run-Time

1 Close the Report Designer.

2 Select the Standard tab of the Delphi component palette.

3 Add a Button component to the form.

4 Configure the Button component:

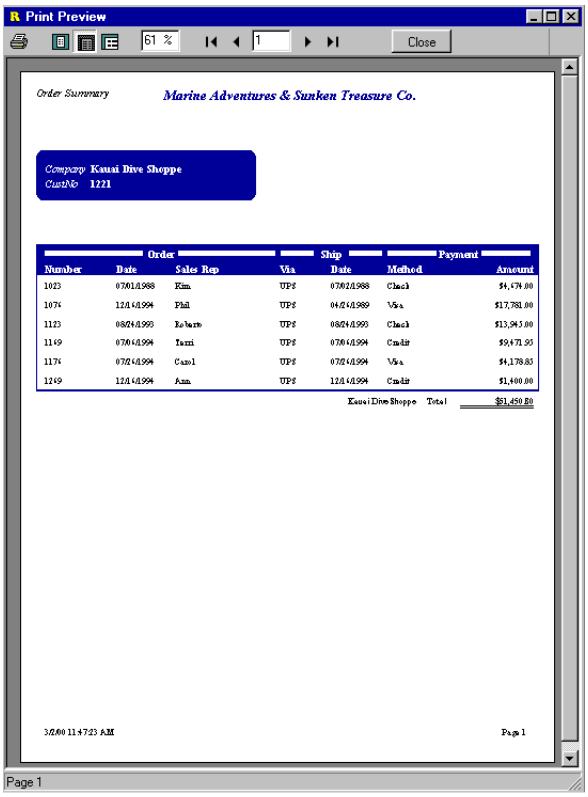
Name	btnPreview
Caption	Preview

5 Add the following code to the OnClick event handler of the button:

```
rbOrderSummary.Print;
```

6 Select File | Save from the Delphi main menu.

7 Run the Project. Click the Preview button. The report should look like this:

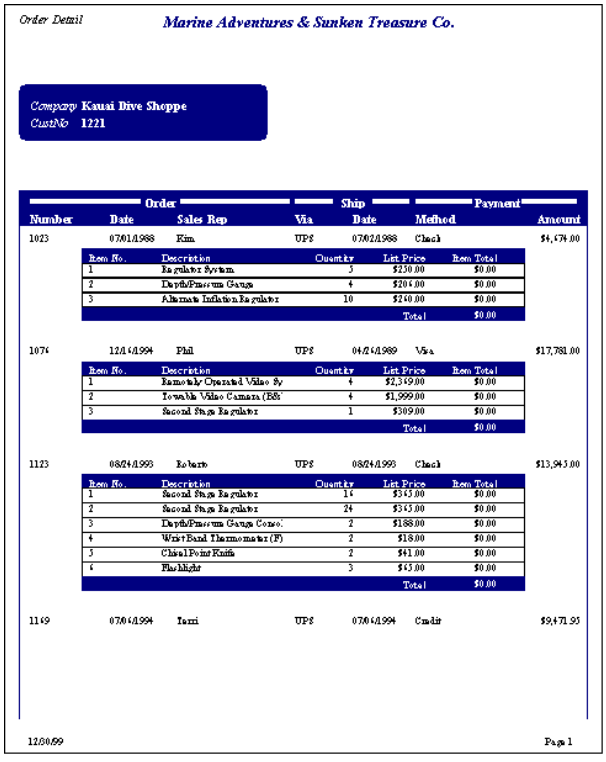


Master→ Detail→ Detail Report

Overview

This tutorial will show you how to do the following:

- Configure Delphi data access objects for use in a master/detail/detail report
- Configure Delphi data access objects for use as a lookup table
- Use a subreport to print detail data
- Perform calculations
- Define a group break


Order Detail

Marine Adventures & Sunken Treasure Co.

Company Kauai Dive Shoppe
CustNo 1221

Order			Ship		Payment	
Number	Date	Sales Rep	Via	Date	Method	Amount
1023	07/01/1988	Kim	UPS	07/02/1988	Check	$4,474.00

Item No.	Description	Quantity	List Price	Item Total
1	Regulator System	3	$230.00	$0.00
2	Depth/Pressure Gauge	4	$20 4.00	$0.00
3	Alternate Inflation Regulator	10	$240.00	$0.00
			Total	$0.00

1076	12/14/1994	Phil	UPS	04/24/1989	Visa	$17,781.00

Item No.	Description	Quantity	List Price	Item Total
1	Remotely Operated Video Sy	4	$2,349.00	$0.00
2	Towable Video Camera (Edit	4	$1,999.00	$0.00
3	Second Stage Regulator	1	$309.00	$0.00
			Total	$0.00

1123	08/24/1993	Roberto	UPS	08/24/1993	Check	$13,943.00

Item No.	Description	Quantity	List Price	Item Total
1	Second Stage Regulator	14	$345.00	$0.00
2	Second Stage Regulator	24	$345.00	$0.00
3	Depth/Pressure Gauge Compo.	2	$188.00	$0.00
4	Wrist Band Thermometer (F)	2	$18.00	$0.00
5	Chisel Point Knife	2	$41.00	$0.00
6	Flashlight	3	$45.00	$0.00
			Total	$0.00

1149	07/04/1994	Ilani	UPS	07/04/1994	Credit	$9,471.93

12/30/99 Page 1

Create a New Application

Note: This tutorial builds upon the Master-Detail report created in the previous section. You can either complete this tutorial or copy the rbMD form from the RBuilder\Tutorials directory.

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Close the new form and unit without saving.

3 Select File | Open from the Delphi main menu. Locate the rbMD.pas unit and open it. Change the form name to 'frmMasterDetailDetail'.

4 Select File | Save As from the Delphi menu and save the form under the name rbMDD in the My RB Tutorials directory (established on page 181).

Note: It is important to name the form and save the form using the names given above because the report will be used in later tutorials.

5 Select View | Project Manager from the Delphi main menu.

6 Right-click over the project name in the Project Manager (usually Project1.exe) and select Add. Add the rbMDD form to the project.

7 Right-click over the project name once again and select the Save menu option.

8 Save the project under the name rbMDDProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Update the Report Name

1 Select the report component.

2 Set the Name to rbOrderDetail.

3 Double-click the Preview button on your Delphi form.

4 In the OnClick event handler change the report name to:

```
rbOrderDetail.Print;
```

Add the Table, DataSource and DataPipeline for the Items Detail Table

1 Add a Table component to the form.

2 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblItem
TableName	items.db

3 Add a DataSource component to the form.

4 Configure the DataSource component:

DataSet	tblItem
Name	dsItem

5 Select the RBuilder tab of the Delphi component palette.

6 Add a DBPipeline  component to the form.

7 Configure the DBPipeline component:

DataSource	dsItem
Name	plItem

Define the Relationship Between the Order and Items Table

1 Select the Item table component.

2 Set the MasterSource property to dsOrder.

3 Select the MasterFields property and press the "..." button. This displays the Field Link Designer.

4 Click on the Available Indexes drop-down list and select ByOrderNo.

5 Select the OrderNo item in the Detail Fields list.

6 Select the OrderNo item in the Master Fields list.

7 Click the Add button. A new item (OrderNo → OrderNo) will be added to the Joined Fields list.

8 Press the OK button to close the dialog.

Note: The IndexName property is now set to ByOrderNo. This property was set automatically when we selected the ByOrderNo index in step 4 above. To establish a master →detail relationship, the detail table must be indexed on the linking field value or values. You can think of an Index as defining the sort order for the table. Thus, if we need to link the Items table to the Order table based on OrderNo, the Items table must have an index (i.e. sort order) based upon OrderNo.

Create the Table, DataSource and DataPipeline for the Parts Lookup Table

1 Add a Table component to your form.

2 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblPart
TableName	parts.db

3 Add a DataSource component to the form.

4 Configure the DataSource component:

DataSet	tblPart
Name	dsPart

5 Select the RBuilder tab of the Delphi component palette.

6 Add a DBPipeline  component to the form.

7 Configure the DBPipeline component:

DataSource	dsPart
Name	plPart

Define the Relationship Between the Item and Parts Table

1 Select the Part table component.

2 Set the MasterSource property to dsItem.

3 Launch the Field Link Designer from the MasterFields property.

4 Select the PartNo in the Detail Fields list.

5 Select the PartNo in the Master Fields list.

6 Click the Add button. A new item (PartNo PartNo) will be added to the Joined Fields list.→

7 Click the OK button to close the dialog.

Note: In this example you will notice the fields being linked have the same name. This is a good standard to follow when designing a database. However, it is not a requirement for linking fields.

8 Select File | Save from the Delphi main menu.

Update the Report Title

1 Access the Report Designer.

2 Locate the Order Summary label in the upper left corner of the header band.

3 Set the caption to Order Detail.

Create SubReport2 and Connect it to the Data

1 Click the 'Subreport1' tab.

2 Right-click over the vertical line component located at the extreme left of the detail band. Select the StretchWithParent and ReprintOnOverflow menu options.

3 Right-click over the vertical line component at the far right of the detail band. Select the StretchWithParent and ReprintOnOverflow menu options.

Note: The detail band will contain a child-type subreport that will stretch to print all of the items for the current order. By setting StretchWithParent to True, we are saying “Whatever the height of the detail band is in the report, resize the line to match it.” Sometimes the items for an order will overflow onto an additional page. By setting ReprintOnOverFlow to True we are saying “When the detail band overflows onto an additional page, make sure to reprint the line.”

4 Set the height of the detail band to 0.4687.

5 Place a subreport component just below the DBText components in the detail band.

6 Right-click over the subreport and set Parent-Width to false, then set the position:

Left	0.875
Top	0.2708
Width	6.25

7 Use the drop-down list at the upper left corner of the Report Designer to set the subreport's DataPipeline to plItem.

8 Select File | Save from the Delphi main menu.

layout check

Remove the Title and Summary Band

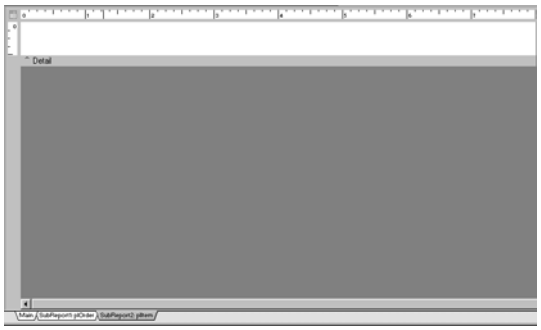
1 Click the 'SubReport2' tab.

2 Select Report | Title from the Report Designer menu to remove the title band.

3 Select Report | Summary from the Report Designer menu to remove the summary band.

Note: Within the context of a child subreport, the title band would print once at the beginning of the report and the summary band would print once at the end. This would be fine for our purposes except that all of the items for a particular order may not fit on a single page, and so the subreport may overflow onto additional pages. When this happens, we want to reprint the header labels for the item fields. The ReprintOnSubsequent property of Group Header bands will work nicely for this requirement, so we will be using a group header and group footer band instead of the title and summary band.

layout check



Create a Group on the 'OrderNo' Field

1 Select Report | Groups from the Report Designer main menu. The Groups dialog will be displayed.

2 Select plItem.OrderNo from the drop-down list at the top of the dialog and click the Add button. A new entry entitled Group[0]: plItem.OrderNo will be added to the list.

3 Click the OK button. You should see two new bands: one above and one below the detail band. These bands have no height; thus, no white space appears above the dividers.

Note: We created a group on the OrderNo field because this is the linking field between the Order and Item tables. This field value will never change during the course of the subreport's generation; thus, the group will never 'break'. The effect will be that the group header prints once at the beginning and once for all subsequent pages. The group footer will print once at the end. You can use this 'no break' technique whenever you need a separation header in a subreport.

4 Set the height of the new bands:

Group Header Band	0.30
Group Footer Band	0.45

5 Select File | Save from the Delphi main menu.

layout check

Lay Out the Group Header Band for the SubReport

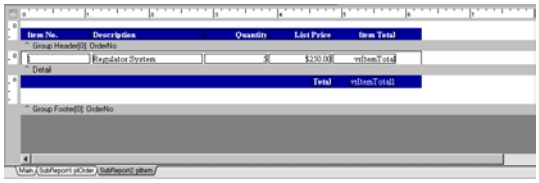
1 Place a shape component in the upper left corner of the group header band.

2 Configure the shape component:

Left	0
Width	6.25
Height	0.2
Fill Color	Navy
Line Color	Navy

3 Hold down the Ctrl key and use the down arrow key to move the shape until the bottom is flush with the bottom of the band.

layout check



Begin Laying Out the Items Information Components for the SubReport

1 Select View | Toolbars | Data Tree from the Report Designer menu. Position the Data Tree to the left of the Report Designer.

2 Click the Layout tab at the bottom of the Data Tree and configure the drag-and-drop settings as follows:

Create	All
Style	Tabular
Label Font Name	Times New Roman
Label Font Style	Bold
Label Font Size	10 point
Label Font Color	White
Field Font Name	Times New Roman
Field Font Style	Regular
Field Font Size	10 point
Field Font Color	Black

3 Click on the Data tab and then select the plItem data pipeline from the list at the top.

4 Select the ItemNo field.

5 Drag the selection into the group header band, positioning the mouse just inside the upper left corner of the shape. Release the mouse button. A DBText component and corresponding label will be created.

6 Hold down the Ctrl key and use the arrow keys to position the selection:

Left	0.1042
Top	0.1354

7 Left justify the components.

8 Set the label caption to Item No.

9 Select the plPart data pipeline in the Data Tree.

10 Select the Description field.

11 Drag the selection into the group header band, positioning the mouse to the immediate right of the Item No. label. Release the mouse button. A DBText component and corresponding label will be created.

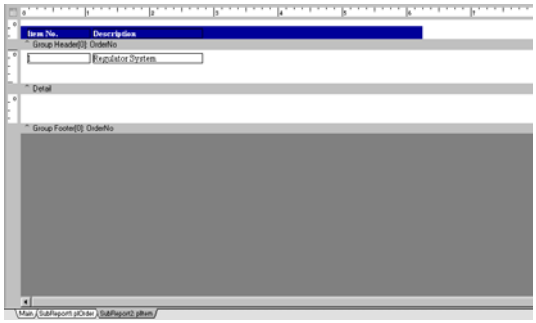
12 Hold down the Ctrl key and use the arrow keys to position the selection:

Left	1.125
Top	0.1354

13 Hold down the Shift key and use the left arrow key to decrease the width of both components to 1.7083.

14 Set the top of both DBText components in the detail band to 1.6667.

layout check



Complete the Items Information Layout

1 Select the Qty field of the plItem data pipeline in the Data Tree.

2 Drag the selection to the immediate right of the Description label.

3 Position the selection:

Left	2.8646
Top	0.1354

4 Right justify the components.

5 Set the label caption to Quantity.

6 Align the top of the new DBText with the top of the existing DBText components.

7 Select the List Price field of the plPart data pipeline.

8 Drag the selection to the immediate right of the Quantity label.

9 Position the selection:

Left	3.8646
Top	0.1354

10 Right justify the components.

11 Set the label caption to List Price.

12 Set the Display Format of the DBText component to $#,0.00;($#,0.00).

13 Align the top of the new DBText component with the top of the existing DBText components.

14 Place a label component to the right of the List Price label.

15 Configure the label:

AutoSize	False
Caption	Item Total
Font Name	Times New Roman
Font Size	10 point
Font Style	Bold
Font Color	White
Left	4.8646
Top	0.1354
Width	0.9792
Text Alignment	Right justified

16 Place a variable component in the detail band just below the Item Total label.

17 Configure the variable component:

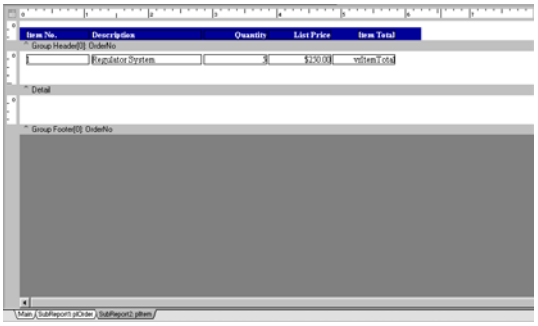
Name	vrItemTotal
DataType	dtCurrency
Font Name	Times New Roman
Font Size	10 point
Font Style	Regular
Font Color	Black
Text Alignment	Right
DisplayFormat	$#,0.00;($#,0.00)
Width	0.9792

18 Select the Item Total label, then Shift-click the vrItemTotal variable. Click the Align Right icon  on the Align or Space toolbar.

19 Align the top of the variable component with the top of the DBText components.

20 Select File | Save from the Delphi main menu.

layout check



Complete the Detail Band Layout for the SubReport

1 Place a shape component in the upper left corner of the detail band.

2 Configure the shape component:

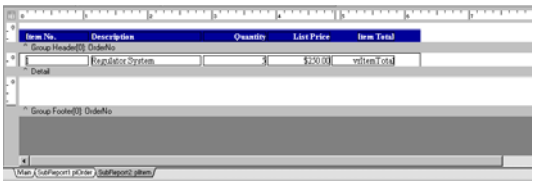
Left	0.0
Top	0.0
Width	6.25
Height	0.2187
Line Color	Black
Fill Color	None

3 Right-click over the shape and choose the Send To Back menu option.

4 Set the height of the detail band to 0.2083.

Note: Notice how the bottom of the shape extends one screen pixel beyond the end of the detail band. This technique allows the shapes to overlap when rendered on the page, creating the effect of a single line between detail bands.

layout check



Lay Out the Group Footer Band for the SubReport

1 Select the shape component in the group header band.

2 Copy and paste the selection. Drag the new shape into the group footer band and position it at the upper left corner:

Left	0.0
Top	0.0

3 Select the Item Total label in the group header band.

4 Copy and paste the selection. Drag the new label into the group footer band.

5 Set the label caption to Total.

6 Select the List Price label in the group header band, then Shift-click the Total label. Click the Right Align icon on the Align or Space toolbar.

7 Select the shape in the group footer band, then Shift-click to select the Total label. Click the Align Center icon on the Align or Space toolbar.

8 Select the vrItemTotal variable in the detail band.

9 Copy and paste the selection. Drag the new variable into the group footer band.

Note: This technique of copy/paste can be much more efficient than creating a new component because the DisplayFormat and Font proper values are returned.

10 Configure the variable:

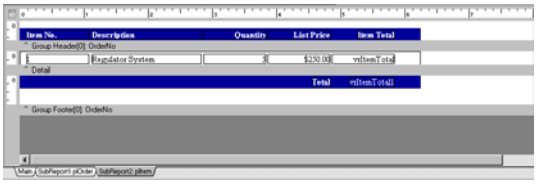
Name	vrOrderTotal
Font Color	White

11 Use the Align or Space toolbar to Right Align the vrOrderTotal variable with the vrItemTotal variable.

12 Align the top the vrOrderTotal variable with the top of the Total label.

13 Select File | Save from the Delphi main menu.

layout check



Code the Calculations for the Totals

1 Select the vrItemTotal variable in the detail band.

2 Select the Events tab of the Object Inspector.

3 Double-click on the OnCalc event. An event handler shell will be generated in your Delphi form.

4 Add the following code to this event handler:

```
Value := plItem['Qty']*plPart['ListPrice'];
```

Note: This event handler retrieves the quantity and list price of the item and returns the total amount via the Value parameter. The result will become the value of the variable component. The OnCalc event will fire once for each item record.

5 Return to the Report Designer and select the vrOrderTotal variable in the group footer band.

6 Code the OnCalc event:

```
Value := Value + vrItemTotal.Value;
```

Note: This event handler retrieves the current total for the item and adds it to the running total for the order. This OnCalc event will also fire once for each item record.

7 Return to the Report Designer, right-click over the vrOrderTotal variable, and select the Timing... menu option.

8 Select 'GroupEnd' from the Reset On drop-down list. Select 'Group0: OrderNo' from the Group drop-down list.

Note: The Timing dialog allows you to control when the OnCalc event will fire and when the variable value will be reset. For this total, we want to calculate the value each time a record is traversed and we want to reset the value after the group footer has completed printing.

9 Select Project | Compile rbMDDProj. Fix any compilation problems.

10 Select File | Save from the Delphi main menu.

Complete the Layout for SubReport1

1 Click the 'SubReport1' tab.

2 Right-click over the Orderno DBText component located on the far left side of the detail band. Set the ReprintOnOverFlow menu option to True.

Note: OverFlow occurs when the detail band runs out of page space on the current page and must complete printing on the next page. For this report we want to print the ItemNo followed by 'Continued...' at the top of the page when the orders for a given customer overflow onto an additional page.

3 Place a label in the detail band and configure it:

Caption	Continued...
Name	lblContinued
Visible	False
Font Name	Times New Roman
Font Size	10
Font Style	Regular
Font Color	Black
Text Alignment	Left justified

4 Select the SaleDate DBText component, then Shift-click the Continued label.

5 Click the Align Top and Align Left icons on the Align or Space toolbar.

6 Add the following code to OnPrint event of the Continued label:

```
lblContinued.Visible :=
  ppChildReport1.Detail.OverFlow;
```

7 Select Project | Compile rbMDDProj. Fix any compilation problems.

8 Select File | Save from the Delphi main menu.

Note: The OverFlow property of the detail band will only be True when the orders for the current customer do not fit on the page and therefore must overflow onto an additional page. When this happens, the label will be visible.

Convert the Title Band to a Group Header Band

1 Select Report | Groups and create a group on plOrder.CustNo. Uncheck the 'Keep group together' option.

2 Set the height of the group header band to 0.5208.

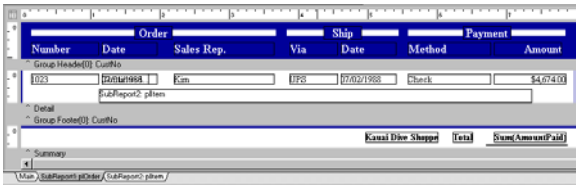
3 Right click over the shape in the title band and set ParentHeight and ParentWidth to False.

4 Select all of the components in the title band, including the shape.

5 Drag the selection into the group header band.

6 Select Report | Title from the Report Designer main menu to remove the title band.

layout check



7 Preview the completed report.

Note: You should be able to get a general idea of how things look. However, you will need to preview the report at run-time to see the results of the calculations and other event handlers.

Preview the Report at Run-Time

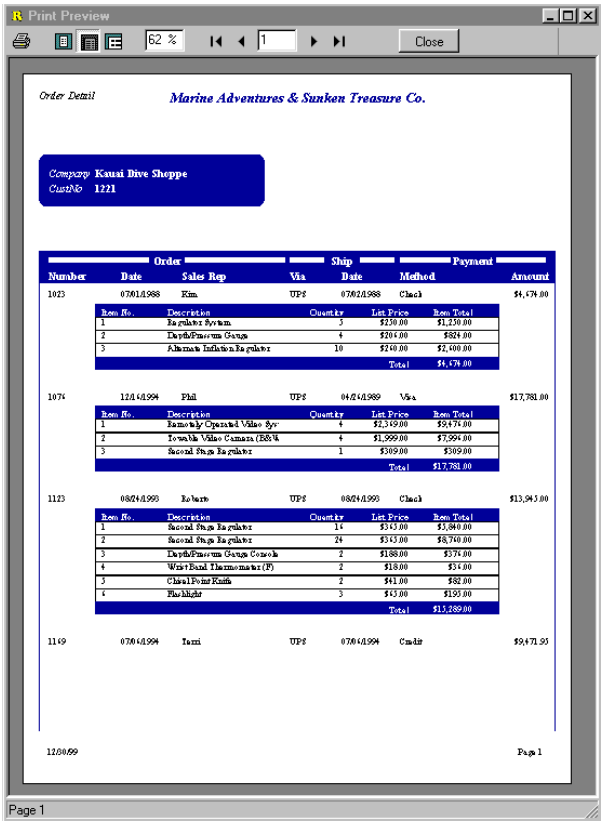
1 Close the Report Designer.

2 Select Project | Compile rbMDDProj. Fix any compilation problems.

3 Select File | Save from the Delphi main menu.

4 Run the project.

5 Preview the completed report. The report should look like this:

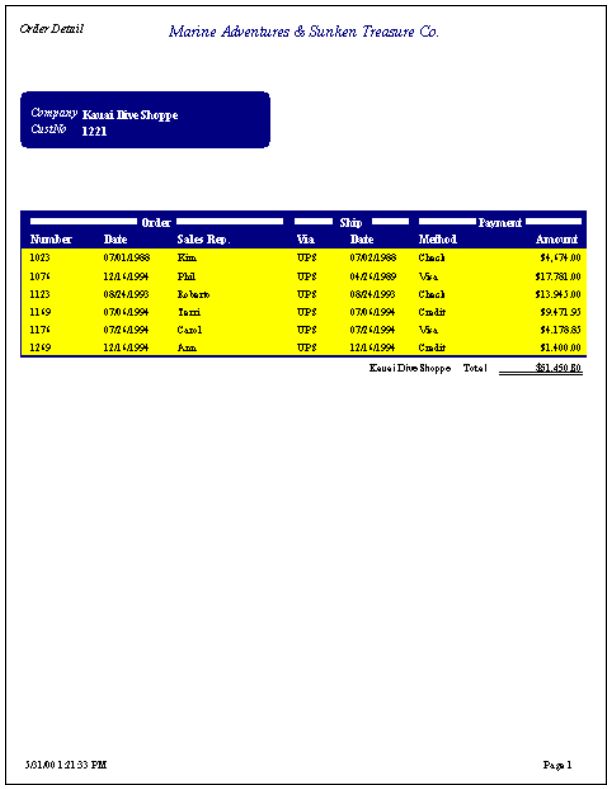


Interactive Previewing with Drill-Down Subreports

Overview

This tutorial will show you how to do the following:

- Use the drill-down feature of child-type subreports



Create a New Application

Note: This tutorial builds upon the master-detail-detail report created in the previous section. You can either complete the previous tutorial or copy the rbMDD form from the RBuilder\Tutorials directory.

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Close the new form and unit without saving.

3 Select File | Open from the Delphi menu. Locate the rbMDD.pas unit and open it.

4 Change the form name to 'frmDrillDownSubreport'.

5 Select File | Save As from the Delphi menu and save the form under the name rbDrillD in the My RB Tutorials directory (established on page 181).

6 Select View | Project Manager from the Delphi main menu.

7 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Add menu option. Add the rbDrillD form to the project.

8 Right-click over the project name in the Project Manager and select the Save menu option.

9 Save the project under the name rbDDProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Invoke the Report Designer and Configure the Drill-Down

1 Double-click on the Report component to display he Report Designer.

2 Size and move the Report Designer window so that the Object Inspector is visible.

3 Select the PageCount System Variable in the footer band. Set the variable type to PageNoDesc.

4 Select Report | Pass Setting | One Pass from the Report Designer main menu.

Note: Drill-down reports should always be set to one-pass. Otherwise, the entire report will be generated each time the user expands the subreport.

5 Click the 'SubReport1' tab.

6 Place a shape in the left side of the detail band.

7 Configure the shape:

Name	shpClickMe
ParentWidth	True
Top	0
Height	0.25
Fill Color	Yellow
Line Color	Yellow

8 Right-click over the shape and select the Send to Back menu option. The shape should appear behind the other components.

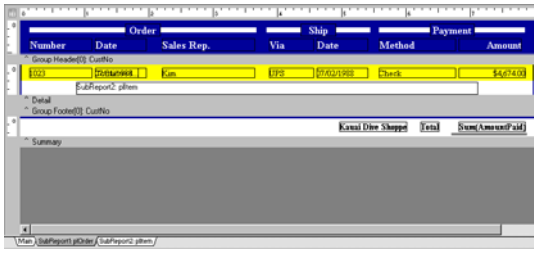
9 Right-click over the Item subreport and select DrillDown.

10 Select the shpClickMe from the drop-down list and click OK.

Note: By specifying the DrillDown option of the subreport, we've associated the yellow shape with the report. Once we've done this, the subreport will not print until the shape is clicked in the Print Preview Window. The DrillDown... dialog sets the DrillDownComponent property of the SubReport.

11 Select File | Save from the Delphi main menu.

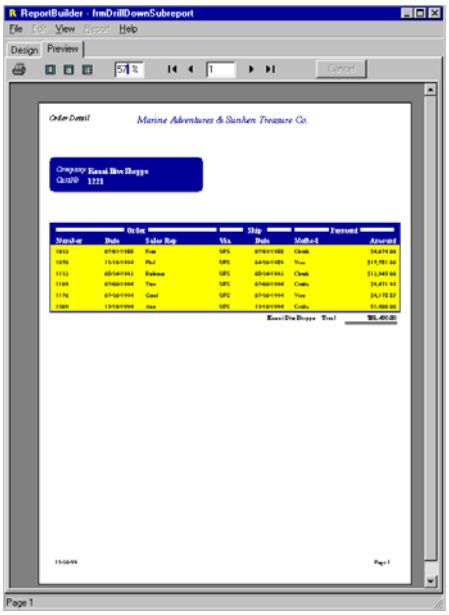
layout check



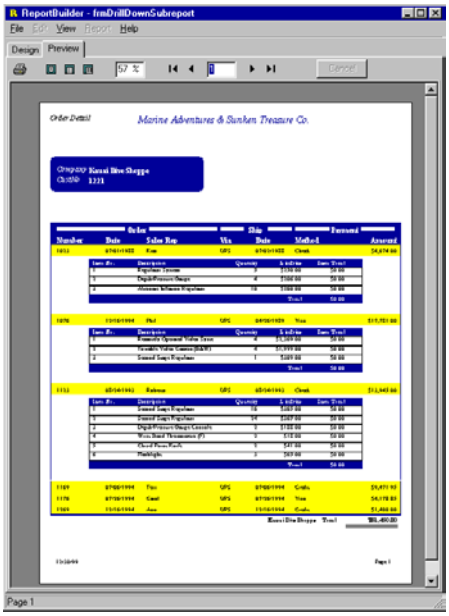
12 Preview the report. The items subreport should not appear. When you move the mouse over the yellow area, the cursor should change to a pointing hand. When you click the yellow area, the subreport should appear. Clicking the same area again should cause the subreport to disappear.

The report should look like this:

Initial View



First three subreports expanded



Hooking Reports Together with Section-Style Subreports

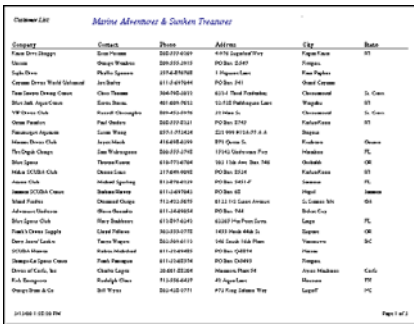
Overview

This tutorial will show you how to do the following:

- Create a single report from two different reports
- Save and load report layouts from a file
- Re-assign event handlers when a report is used in a new form

Title Page

First Page of the Customer List Report



First Page of the Customer List Report

Create a New Application

Note: This tutorial uses the Customer List and the Stock Summary reports created in earlier tutorials. If you have not completed these tutorials, you can use the completed tutorials in the RBuilder\Tutorials directory.

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmSectionSubreports'.

3 Select File | Save As from the Delphi menu and save the form under the name rbSectSR in the My RB Tutorials directory (established on page 181).

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbSSProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Transfer the Customer List Report to the Form

1 Open the form containing the Customer List report (rbCust.pas from "A Simple Report...").

2 Double-click on the Report component to display the Report Designer.

3 Select File | Save As from the Report Designer main menu.

4 Save the report template in the My RB Tutorials directory under the name CustList.

Note: Saving a report layout to file is a quick and easy way to make the report more portable. Later in this tutorial we will open this saved layout in a subreport. Layouts can also be loaded at run-time using the Report.Template.LoadFromFile or LoadFromDatabase method.

5 Close the Report Designer.

6 Select the data access components (tblCustomer, dsCustomer, plCustomer).

7 Copy the components into your clipboard.

8 Select View | Forms from the Delphi main menu. Double-click on frmSectionSubreports to make this form visible.

9 Paste the components onto this form.

10 Place a standard Delphi label on the form, directly above these components.

11 Set the label caption to 'Customer List Data'.

12 Set the Visible property of the label to False.

Note: This invisible label will not be used by the report: it simply serves as a reminder of how these data access components are used. Whenever you have multiple sets of data access components on a form, it is a good idea to label them. An alternate way to accomplish this is to set the Tools | Environment Options | Preferences | Show Component Captions option on, but this method requires that you space the components more widely than we find preferable.

13 Close the rbCust unit.

Transfer the Stock Summary Report to the Form

1 Open the form containing the Stock Summary report (rbStock.pas from the "Groups, Calculations, ..." tutorial).

2 Double-click on the Report component to display the Report Designer.

3 Select File | Save As from the Report Designer main menu.

4 Save the report template in the My RB Tutorials directory under the name StockSum.

5 Close the Report Designer.

6 Select the data access components (qryStock, dsStock, plStock).

7 Copy these components into your clipboard.

8 Select View | Forms from the Delphi main menu. Double-click on frmSectionSub to make this form visible.

9 Paste the components into the form and drag them to a position below the Customer List data access components.

10 Place a standard Delphi label on the form, directly above these components.

11 Set the Caption to 'Stock Summary Data'.

12 Set the Visible property of the label to False.

13 Close the rbStock unit.

14 Select File | Save from the Delphi main menu.

Create and Configure the Main Report

1 Add a Report component to the form.

2 Name the Report component rbSectSub.

3 Double-click on the Report component to display the Report Designer.

4 Size and move the Report Designer window so that the Object Inspector is visible.

5 Select Report | Header from the Report Designer menu. The header band will be removed.

6 Select Report | Footer. The footer band will be removed.

layout check

Create the Customer List SubReport

1 Place a SubReport component in the detail band.

2 Position the subreport so that it is flush with the top of the detail band.

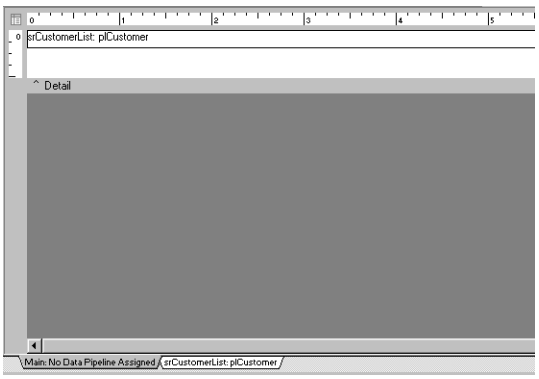
3 Right-click over the subreport and select the Section menu option.

Note: When a subreport is created, it defaults to a PrintBehavior of pbChild. Child subreports print within the context of the parent band, much like a stretching memo. Section subreports generate entire pages, creating a whole section within the parent report.

4 Name the subreport srCustomerList.

Note: Whenever you set the name of a subreport, the text in the subreport component and the text in the tab at the bottom of the Report Designer is updated. The caption is also displayed in the Report Outline section of the Report Tree. When you have several subreports, it is a good idea to name them so it is easier to keep track of them.

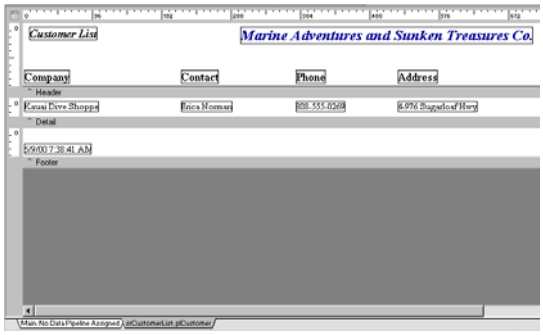
layout check



5 Click the 'srCustomerList' tab.

6 Select File | Load SubReport from the Report Designer main menu and open the CustList.rtm file. The Customer List report will appear in the Report Designer.

layout check



7 Click the Main tab at the bottom of the Report Designer to return to the layout of the main report.

8 Select File | Save from the Delphi main menu.

Create the Stock Summary SubReport

1 Place a second SubReport component in the detail band.

2 Position it so that the top is flush with the bottom of the Customer List subreport.

Note: This step is not necessary: it just helps to create a clean layout. The print order of the subreports is not determined by their top to bottom order within the band. Print order is actually determined by the layering of the components (Send to Back/ Bring to Front order). The report at the back is printed first; the report at the front is printed last. You can quickly determine the layering of subreports via the Report Tree. You can also use the Report Tree to change the layering.

3 Right-click over the subreport and select the Section menu option.

4 Name the subreport srStockSummary.

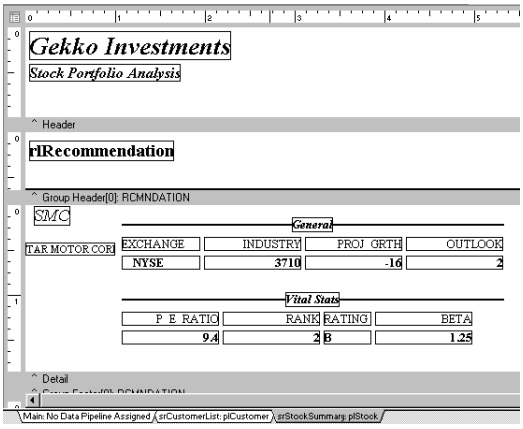
Note: We need to set the name of this subreport because we will be referring to this component by name in an event handler later in the tutorial.

layout check

5 Click the 'srStockSummary' tab.

6 Select File | Load SubReport from the Report Designer main menu and open the StockSum.rtm file. The Stock Summary report will appear in the Report Designer. Ignore any messages here regarding 'Invalid property values', as these are related to the event handlers formerly assigned to the report.

layout check



7 Click the 'Main' tab at the bottom of the Report Designer.

8 Select File | Save from the Delphi main menu.

Preview the Report at Run-Time

1 Close the Report Designer.

2 Select the Standard tab of the Delphi component palette.

3 Add a Button component to the form.

4 Configure the Button component:

Name	btnPreview
Caption	Preview

5 Put the following code in the OnClick event handler of the button:

```
rbSectSub.Print;
```

6 Select File | Save from the Delphi main menu.

7 Run the Project.

8 Click the Preview button. The report should be displayed in the Print Preview form. The first three pages should contain the Customer List report; the last thirty-nine pages should contain the Stock Summary report.

Note: The calculations for the Stock Summary report are not correct, and the color-coding for this report is gone. This is because we have not created the event handlers for this report. We will complete this task in the next section.

Copy the Event Handlers from the Stock Summary Report

1 Re-open the form containing the Stock Summary report (rbStock.pas).

2 Locate the event handler declarations in the form declaration at the top of the unit.

3 Copy the following event handler declarations into your clipboard:

```
procedure  ppGroupHeaderBand1BeforeGenerate;
procedure ppDetailBand1BeforeGenerate;
procedure vrBuyTotalCalc;
procedure vrHoldTotalCalc;
procedure vrSellTotalCalc;
```

4 Locate the form class declaration at the top of the rbSectSR unit.

5 Paste the event handler declaration immediately above the private section of the form class declaration.

Note: You may have noticed that the Delphi Form Designer automatically adds declarations to the mysteriously unlabeled section at the top of your form class declaration. This is actually the published section of the form class. Declarations for all of the event handlers and components within a form are placed in the published section in order to facilitate the streaming logic used to load and save forms to dfm files. By pasting these declarations into the published section of the form (and pasting the corresponding implementations in the unit), we make these event handlers assignable from the Object Inspector.

6 Return to the rbStock unit and copy the 'FRecommendation' variable from the private section of the form class declaration.

7 Return to the rbSectSR unit and paste this variable into the private section of the form class declaration.

8 Return to the rbStock unit.

9 Scroll down to the implementation section of the unit and copy all of the event handlers, save the TfrmStockSummary.btnPreviewClick event.

10 Return to the rbSectSR unit and paste the event handlers in the implementation section of the unit.

11 Select File | Save from the Delphi main menu.

12 Return to the rbStock unit. Right-click over the code editor and select Close Page to close this unit.

Re-attach the Event Handlers to the Stock Summary Subreport

1 Scroll to the top of the rbSectSR unit, double-click the class name TfrmSectionSubreports, and copy it into the clipboard.

2 Scroll down to the implementation section and Replace the existing classname (TfrmStockSummary) for each event handler with the new class name (TfrmSectionSubreports).

3 Double-click on the rbSectSub report component to display the Report Designer.

4 Click on the srStockSummary tab at the bottom of the Report Designer.

5 Click in the white space of the group header band.

6 Select the Events tab of the Object Inspector.

7 Select the BeforeGenerate event, then expand the drop-down list of event handlers and select this routine:

```
ppGroupHeaderBand1BeforeGenerate
```

8 Select the AfterGenerate event to make sure the assignment was successful.

9 Click in the white space of the detail band.

10 Select the BeforeGenerate event, then expand the drop-down list of event handlers and select this routine:

```
ppGroupHeaderBand1BeforeGenerate
```

11 Select the AfterGenerate event to make sure the assignment was successful.

12 Click in the white space of the group header band once again and double-click on the BeforeGenerate event to activate the code editor. Change the 'if' statement on the last line of the event handler to refer to srStockSummay instead of rbStockSum. The code should read:

```
if srStockSummary.Report.Groups[0].FirstPage
then
```

Note: The original event handler referred to the rbStockSum report component. This event handler has to be changed to refer to the Report component within the srStock subreport. A subreport component actually consists of two components: the subreport component and the report component contained within it. The subreport component is used to position the subreport within the parent report and to set various properties

related to print behavior. The report component is accessible via the Report property and contains the report itself. This component is no different than the report component you work with on a Delphi form.

13 Scroll down to the summary band and reconnect the Variable components to their corresponding OnCalc events:

Component	Event
vrBuyTotal	vrBuyTotalCalc
vrHoldTotal	vrHoldTotalCalc
vrSellTotal	vrSellTotalCalc

Note: This tutorial illustrates how to reconnect the event handlers of the Stock Summary report so that you will better understand the situation. If we had copied these event handlers into the SectSub unit before we loaded the Stock Summary report into the subreport, the event handlers would be reconnected automatically by Delphi's object streaming logic. (You may recall that an error message stating that the event handlers could not be found was displayed when we loaded the report layout.) It is important to know that event handlers are reconnected based on their procedure name. When you want event handlers to reconnect automatically, they must have the same names as those saved in the report layout.

14 Select File | Save from the Delphi main menu.

15 Run the Project. The Stock Summary report should now be color-coded and the totals should calculate properly.

Add a Title Page to the Report

1 Double-click on the rbSectSub report component to display the Report Designer.

2 Click the 'Main' tab.

3 Select Report | Title from the Report Designer main menu. A title band will be created.

4 Right-click over the title band and select the Position... menu option.

5 Set the height to 5.8125.

Begin Laying Out the Title Band

1 Place a label component in the upper left corner of the title band.

2 Configure the label:

Caption	Title Page
Font Name	Times New Roman
Font Size	12
Font Style	Bold
Font Color	Black
Text Alignment	Left

3 Place a label component near the middle of the title band.

4 Configure the label:

Caption	Marine Adventures & Sunken Treasures Co.
Font Name	Times New Roman
Font Size	20
Font Style	Bold & Italic
Text Alignment	Center
Left	1.3542
Top	1.9166

5 Place a label component just below the previous label.

6 Configure the label:

Caption	Annual Report for 1997
Font Name	Times New Roman
Font Size	20
Font Style	Bold & Italic
Text Alignment	Center
Left	2.5729
Top	2.3021

7 Place a label component just below the previous label.

8 Configure the label:

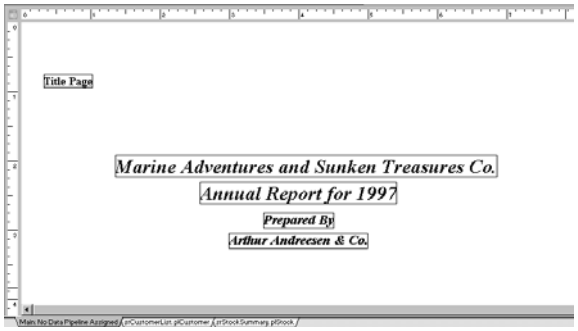
Caption	Prepared By
Font Name	Times New Roman
Font Size	14
Font Style	Bold & Italic
Text Alignment	Center
Left	3.4896
Top	2.75

9 Place a label component just below the previous label.

10 Configure the label:

Caption	Arthur Andreesen & Co.
Font Name	Times New Roman
Font Size	14
Font Style	Bold & Italic
Text Alignment	Center
Left	2.9896
Top	3.0416

layout check



Complete the Title Band Layout

1 Place a label component just below the previous label.

2 Configure the label:

Caption	Table of Contents
Font Name	Times New Roman
Font Size	20
Font Style	Bold & Italic
Text Alignment	Center
Left	2.9375
Top	3.8646

3 Place a label component just below the previous label.

4 Configure the label:

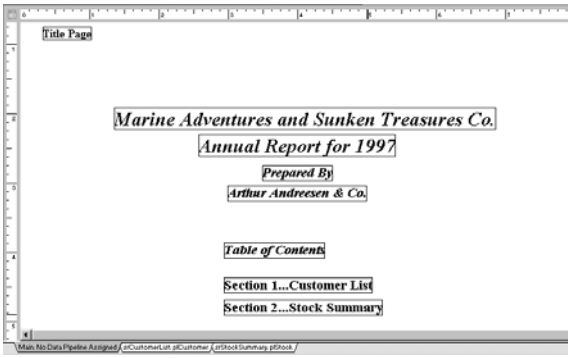
Caption	Section 1.....Customer
Font Name	Times New Roman
Font Size	14
Font Style	Bold
Text Alignment	Center
Left	2.9375
Top	4.3646

5 Place a label component just below the previous label.

6 Configure the label:

Caption	Section 2.....Stock
Font Name	Times New Roman
Font Size	14
Font Style	Bold
Text Alignment	Center
Left	2.9375
Top	4.6875

layout check

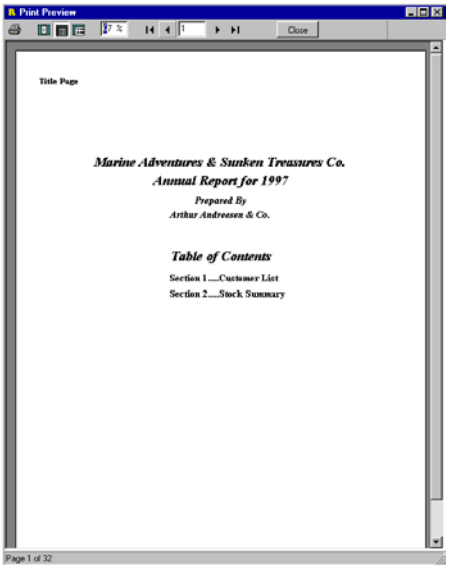


7 Select Project | Compile rbSSProj. Fix any compilation problems.

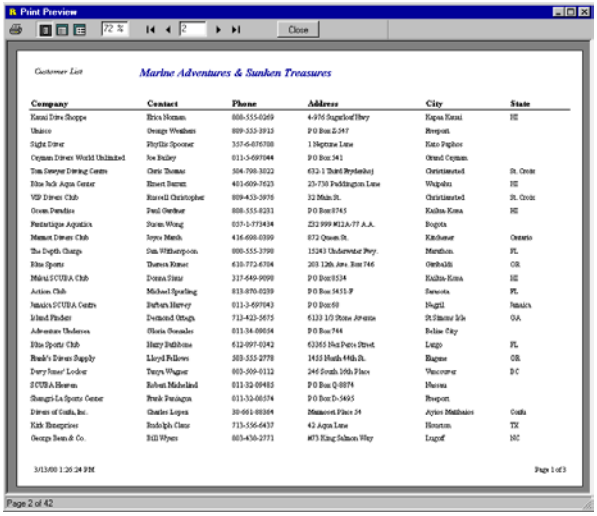
8 Select File | Save from the Delphi main menu.

9 Run the Project. The first page of each section should look like the following:

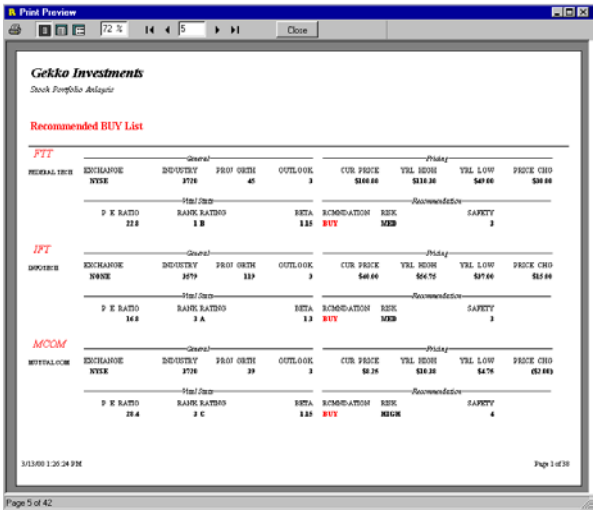
Title Page



First Page of the Customer List Report



First Page of the Stock Summary Report

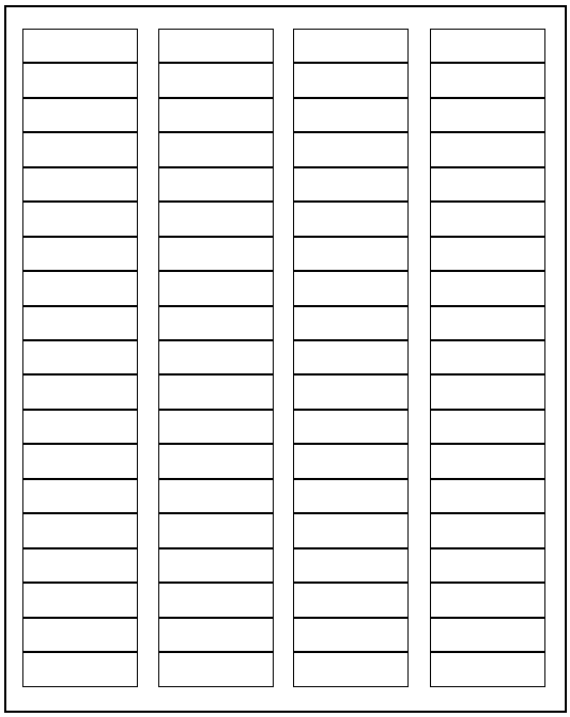


Using Columns to Create Mailing Labels

Overview

This tutorial will show you how to do the following:

- Configure a report to print mailing labels



Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmMailingLabels'.

3 Select File | Save As from the Delphi menu and save the form under the name rbMailL in the My RB Tutorials directory (established on page 181).

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save option.

6 Save the project under the name rbMLProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Invoke the Report Designer and Configure the Page Layout

1 Select the RBuilder tab of the Delphi component palette.

2 Place a report component on the form.

3 Double-click on the Report component to display the Report Designer.

4 Size and move the Report Designer window so that the Object Inspector is visible.

5 Select File | Page Setup from the Report Designer main menu.

6 Select the Layout tab. Enter the following values:

Columns	4
Column Width	1.75
Column Position 1	0.28
Column Position 2	2.34
Column Position 3	4.4
Column Position 4	6.46

7 Select the Margins tab. Enter the following values:

Top	0.4
Bottom	0.49
Left	0.2969
Right	0.2969

8 Click the OK button to close the dialog.

layout check

Configure the Report

1 Click the Select Report icon at the upper left corner of the Report Designer workspace.

2 Set the PageLimit to 1 and the AutoStop property to False in the Object Inspector.

Note: This report is not connected to any data, so it has no basis on which to stop generating pages. In this situation, the AutoStop property defaults to True so that the report will print only one detail band. Because we want to fill a single page with detail bands, we turn AutoStop off and set the PageLimit to one.

3 Select Report | Header from the Report Designer main menu. The header band will be removed from the report.

4 Select Report | Footer from the Report Designer main menu. The footer band will be removed from the report.

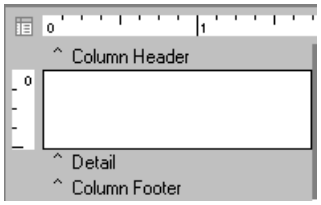
5 Place a shape in the detail band.

6 Configure the shape:

ParentHeight	True
ParentWidth	True

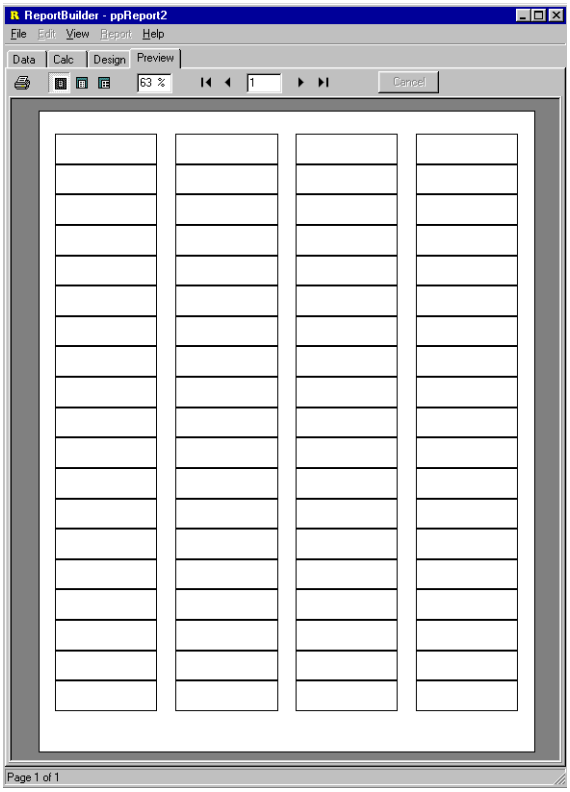
7 Select File | Save from the Delphi main menu.

layout check



8 Preview the report. You should see four columns of shapes. Print this page to the printer.

The labels should look like this when you preview:



9 Take the sheet just printed and place it behind an Avery 5267 Laser label sheet. Hold the sheets up to a bright light so you can see exactly where the lines fall on the labels. Ideally, the edges of the lines should match up with the edges of the labels.

Note: By placing a shape in the detail band, we can see exactly where the label information will print. If the shape does not match up with the labels, we can make adjustments to the margins, column positions, and row spacing of the detail band. It is fairly common for the alignment shape to be off a little, even though the exact measurements are sent to the printer. This is usually due to variation in the label sheets and to the calibration of the printer being used. Generally, you must tweak the report layout until the alignment shape appears in the correct location, then set the Visible property of the shape to False and use the template like any other report.

Create Mailing Labels Via the Label Template Wizard

1 Place another report component on the form and double-click on it to display the designer.

2 Select File | New from the main menu of the designer.

3 Double-click on the Label Templates icon.

4 Make the following selections:

Products	Avery Standard
	5267 - Return Address

5 Place a shape in the detail band.

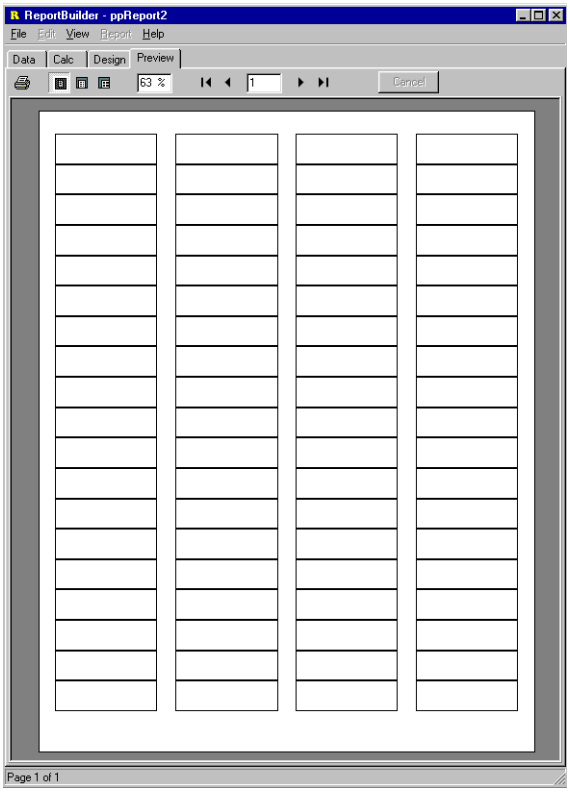
6 Set the ParentHeight and ParentWidth of the shape to True.

7 Click the Select Report icon and use the Object Inspector to set AutoStop to False and PageLimit to 1.

8 Select File | Save from the Delphi main menu.

9 Preview.

The labels should look like this:



Printing to a Text File

Overview

This tutorial will show you how to do the following:

- Print a report to an ASCII data file

Create a New Application

Note: This tutorial builds upon the 'Creating a Report via the Report Wizard' tutorial. You can either complete this tutorial or use the rbViaWiz form located in the RBuilder\Tutorials directory.

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Close the new form and unit without saving.

3 Select File | Open from the Delphi main menu. Locate the rbViaWiz unit in the Tutorials directory and open it.

4 Set the Form name to 'frmPrintToTextFile'.

5 Select File | Save As from the Delphi menu and save the form under the name rbToTxt in the My RB Tutorials directory (established on page 181).

Note: It is important to name the form and save the form's unit using the names given above because the form will be used in later tutorials.

6 Select View | Project Manager from the Delphi main menu.

7 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Add menu option.

8 Add rbToTxt to the project.

9 Right-click over the project name once again and select the Save menu option.

10 Save the project under the name rbTTProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Complete the Report Wizard Tutorial

1 Rename the Report component rbPrintData.

2 Delete the Preview button from the form.

Load the Tabular Style Report Template

1 Double-click on the Report component to display the Report Designer.

2 Select File | Open from the Report Designer main menu.

3 Open the CustTab.rtm file.

Assign User Names to the DBText Components

1 Set the UserName for each of the DBText components based on the field to which they are assigned. Use the Object Inspector to assign the following UserNames:

dbtCompany
dbtCustNo
dbtContact
dbtPhone
dbtFax
dbtCity
dbtState
dbtCountry

Specify the File Name and Format

1 Select File | Print to File Setup from the Report Designer main menu. The Print to File Setup dialog will be displayed.

2 Click the File button.

3 Access the My RB Tutorials directory and enter customer as the file name. Click the Save button to close the file dialog.

4 Select Tab Delimited from the File Type drop-down list.

5 Select the detail band in the Bands list. The Available Controls list will display the components residing in the detail band.

6 Use the arrows to the right of the Available Controls list box to move each of the DBtext components to the Selected Controls list box in the following order:

Company
CustNo
Contact
Phone
Fax
City
State
Country

7 Click the OK button to close the dialog.

Configure the Report to Print to File

1 Click the Select Report icon located at the upper left corner of the Report Designer workspace where the horizontal and vertical rulers meet.

2 Set the Report's AllowPrintToFile property to True in the Object Inspector.

3 Set the Report's DeviceType to TextFile.

4 Select File | Save As from the Report Designer menu and save the modifications to CustTab.rtm.

Soft Code the Template Name

1 Close the Report Designer.

2 Access the Delphi Code Editor.

3 Search the unit for all occurrences of 'rbCustomerList' and change them to 'rbPrintData.'

Soft Code the Text File Name

1 Access the OnClick event for the Print button and change the event handler so that it looks like the code below.

2 Select Project | Compile rbTTProj. Fix any compilation problems.

3 Select File | Save from the Delphi main menu.

Print the Report to File

1 Run the project.

2 Click the Print button. Notice that the Print to File option is selected.

3 Click OK to print to the file.

4 Close the form and return to the Delphi design environment.

5 Select File | Open from the Delphi main menu and set the files of type to txt. Open the Customer.txt file. The file should contain 55 tab-deliminated records.

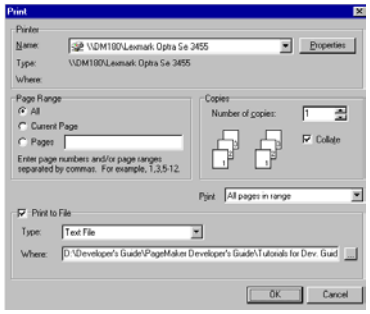
6 Run the project again.

7 Select the Vertical style report and click the Print Button. Notice that there are not any Print to File options in the dialog.

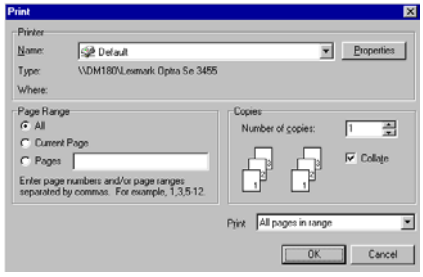
8 Close the dialog and the form.

The print dialog should look like this:

Tabular Report Print Dialog with Print to File Option



Vertical Report Print Dialog without Print to File Option



Code | *Event handler for BuildEmployeeAddress procedure*

```
procedure TfrmPrintToTextFile.btnPrintClick(Sender: TObject);
begin
  {load the template file and print to printer}
   rbPrintData.Template.LoadFromFile;

   if rbTabular.Checked then
     begin
       rbPrintData.AllowPrintToFile := True;
       rbPrintData.TextFileName := FPathName + 'Customer.txt';
       rbPrintData.DeviceType := dtTextFile;
     end
   else
     begin
       rbPrintData.AllowPrintToFile := False;
       rbPrintData.TextFileName := '';
       rbPrintData.DeviceType := dtPrinter;
     end;

   rbPrintData.Print;
end;
```

Printing from a Text File

Overview

This tutorial will show you how to do the following:

- Print a report from data stored in a text file

Create a New Application

Note: This tutorial uses the Customer.txt file created in the previous tutorial. You can either complete this tutorial or use the Customer.txt file located in the RBuilder\Tutorials directory.

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmPrintFromTextFile'.

3 Select File | Save As from the Delphi menu and save the form under the name rbFrmTxt in the My RB Tutorials directory.

Note: It is important to name the form and save the form's unit using the names given above because the report will be used in later tutorials.

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbFTProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create the Text DataPipeline Component

1 Select the RBuilder tab of the Delphi component palette.

2 Place a TextPipeline component on the form.

3 Configure the TextPipeline component:

FileName	Customer.txt
FileType	ftTab
Name	plCustomer

Define the Data Fields for the Text Pipeline

1 Double-click on the TextPipeline component to display the Fields Editor.

2 Add a new field by clicking the Add button.

3 Set the following properties for the field:

FieldName	Company
FieldLength	30

Note: The TextPipeline enables ReportBuilder to use data from a text file as if it were stored in a database. The field lengths specified here are maximum lengths. For example, the actual company names stored in the file may vary in length, but none may exceed 30 characters.

4 Add the following fields:

FieldName	FieldLength
CustNo	10
Contact	20
Phone	15
FAX	15
City	15
State	20
Country	20

5 Close the Fields Editor.

6 Select File | Save from the Delphi main menu.

Create a Report and Connect it to the Data

1 Add a Report component  to the form.

2 Configure the Report component:

DataPipeline	plCustomer
Name	rbCustomerList

Test the Data Connection

1 Double-click on the Report component to display the Report Designer.

2 Place a DBText component in the detail band and set it to AutoSize.

3 Use the two drop down lists at the upper left corner of the Report Designer to set the DataPipeline and DataField properties:

DataPipeline	plCustomer
DataField	Company

Note: The report and Report Designer behave in the same manner whether the report is connected to a DBPipeline, a TextPipeline, or any other type of pipeline, including a custom data pipeline descendant that you or another developer has created.

4 Preview the report. The Company field should be listed.

5 Select the Design tab to return to the design workspace.

Load the Vertical Style Report Template

1 Select the File | Open... from the Report Designer main menu.

2 Open the CustVert.rtm file.

Note: The CustVert.rtm file was created in the Report Wizard tutorial. The report was connected to a DBPipeline named plCustomer. We now have a text pipeline named plCustomer whose fields we have defined with the same names as the fields used from the Customer.db table. When the CustVert.rtm report is loaded, the data is automatically reconnected because the pipeline and field names are identical.

3 Select File | Save from the Delphi main menu.

4 Select the Preview tab to view the report.

5 Select the Design tab to return to design mode.

Soft Code the Text File Name

1 Add the following code to the OnCreate event of the form:

```
procedure
  frmPrintFromTextFile.FormCreate(Sender:
  TObject);
begin
plCustomer.FileName :=
  ExtractFilePath(ParamStr(0)) +
  'Customer.txt';
end;
```

Note: This code will allow you to deploy the application without worrying about which directory the customer.txt file is in. This code assumes that the file is located in the same directory as the application.

Preview the Report at Run-Time

1 Select the Standard tab of the Delphi component palette.

2 Add a Button component to the form.

3 Configure the Button component:

Name	btnPreview
Caption	Preview

4 Add the following code to the OnClick event handler of the button:

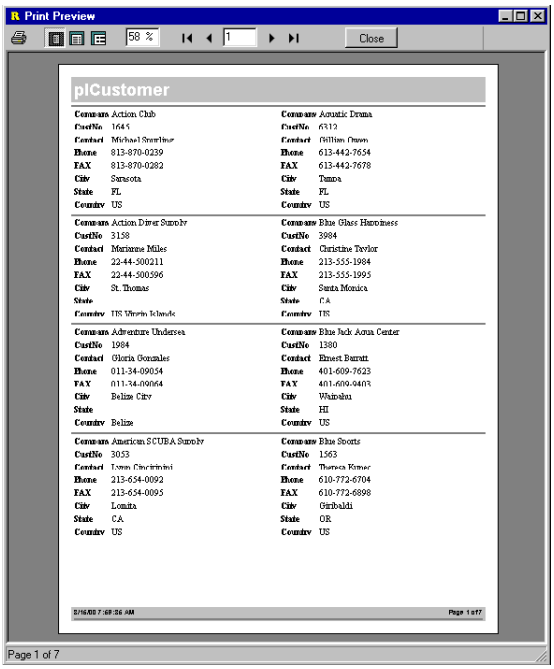
```
rbCustomerList.Print;
```

5 Select Project | Compile rbFTProj. Fix any compilation problems.

6 Select File | Save from the Delphi main menu.

7 Run the project.

8 Click on the Preview button. The report should be displayed in the Print Preview form:

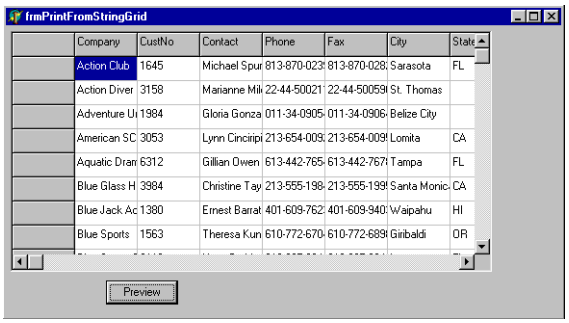


Using the JITPipeline to Print from a StringGrid

Overview

This tutorial will show you how to do the following:

- Print a report from data stored in a string grid or string list



Create a New Application

Note: This tutorial builds upon the Printing from a Text File tutorial. You can either complete the Print from Text File tutorial or use the rbFrmTxt form in the RBuilder\Tutorials directory.

1 Select File | New Application from the Delphi menu. This will create a new project and blank form.

2 Close the blank form and its associated unit without saving.

3 Select File | Open from the Delphi main menu. Open the rbFrmTxt unit in the Tutorials directory.

4 Set the form name to 'frmPrintFromStringGrid'.

5 Select File | Save As from the Delphi menu and save the form under the name rbFrmJIT in the My RB Tutorials directory.

6 Select View | Project Manager from the main menu.

7 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Add menu option.

8 Add rbFrmJIT to the project.

9 Right-click over the project name in the Project Manager and select the Save menu option.

10 Save the project under the name rbFJProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Rename the TextPipeline

1 Rename the TextPipeline plCustomerTxt.

Note: For this example, the text pipeline will be used to load the data from the customer.txt file into a string grid. A JITPipeline will then pull the data from the string grid and provide it to the report.

Create a StringGrid

1 Select the Additional tab of the Delphi component palette and add a StringGrid to the form.

2 Configure the StringGrid:

Left	5
Top	5
Width	500
Height	250
Name	grCustomer

3 Double-click on the Options property in the Object Inspector to expand the options list. Set goColSizing and goRowSelect to True.

Add Code to Load the StringGrid

1 Add the following procedure declaration to the bottom of the interface section for the unit (just below the end of the form class declaration and just above the var declaration).

```
procedure PipeDataToGrid(aPipeline:
↳ TppDataPipeline; aGrid: TStringGrid);
```

2 Copy the PipeDataToGrid routine from form dm0139 in the RBuilder\Demos\1. Reports directory.

3 Paste the code at the bottom of the implementation section of the rbFrmJIT unit.

Note: This procedure is written in a generic manner so that it can be reused as needed. You pass the procedure a data pipeline (TextPipeline, JITPipeline, DBPipeline, etc) and a string grid. The procedure loads all of the data from the pipeline into the string grid. You can store routines such as this one in a common unit when building an application. You can then add this common unit to the uses clause of other units where the utility routines are needed. In this way you can avoid repeating the same routine in several units.

Note: This procedure populates a string grid with pipeline data. The data pipeline is opened and all the data is traversed to count the total number of records. The row size of the grid is set to the number of rows + 1 because the field names are to be stored in the first row (i.e. row 0). The column size of the grid is likewise set to the number of fields + 1 because the first column contains the border for the grid. Next, the field names are loaded into the first row. Finally, the pipeline data is traversed once again and the data is loaded to the grid cells.

4 Change plCustomer to plCustomerTxt in the OnCreate event handler of the form:

```
plCustomerTxt.FileName :=
↳ ExtractFilePath(ParamStr(0)) + 'Customer.txt';
PipeDataToGrid(plCustomerTxt, grCustomer);
```

Note: This code assigns the text file name (assuming the text file is in the same directory as the application) and populates the string grid with data.

5 Select Project | Compile rbFJProj. Fix any compilation problems.

6 Select File | Save from the Delphi main menu.

7 Run the project. The customer data should be displayed in the grid.

Create the JIT DataPipeline Component

1 Select the RBuilder tab of the Delphi component palette.

2 Add a JITPipeline component  to the form. Set the Name property to plCustomer.

3 Set the report component's DataPipeline property to plCustomer.

Define the Data Fields for the JITPipeline

1 Double-click on the JITPipeline component to display the Field Editor.

2 Click the Add button to add a new field.

3 Use the Object Inspector to set the following properties for the field:

FieldName	Company
FieldLength	30

Note: The JITPipeline enables ReportBuilder to access data as if it were stored in a database. The field lengths specified here are maximum lengths.

4 Repeat steps 2 and 3 to add the following fields:

FieldName	FieldLength
CustNo	10
Contact	20
Phone	15
FAX	15
City	15
State	20
Country	20

5 Close the Field Editor.

6 Select the form and add the following code to the OnCreate event (below the call to 'PipeDataToGrid'):

```
plCustomer.InitialIndex := 1;
plCustomer.RecordCount:=grCustomer.RowCount-1;
```

7 Select File | Save from the Delphi main menu.

Add a Function to Return the Grid Field Values

1 Copy the GetGridFieldValue function declaration from form 0139 in the RB\Demos\1. Reports directory.

2 Paste it directly above the PipeDataToGrid declaration.

3 Access form 0139 and copy the GetGridFieldValue routine.

4 Paste the code at the bottom of the implementation section of the rbFrmJIT unit.

Note: This routine returns the data field value for the current record. The field names are stored in the first row of the grid and are used to determine the column number of the given field name. This column number and current row index are used to retrieve the current field value.

Add JITPipeline Event Handlers to Return the Field Values

1 Toggle back to the form and select the plCustomer JIT data pipeline, then click the Events tab on the Object Inspector.

2 Create the following event handler for the OnGetFieldValue:

```
Result := GetGridFieldValue (grCustomer,
  plCustomer.RecordIndex, aFieldName,
  plCustomer.GetFieldDataType(aFieldName));
```

Note: The event handler above simply calls the GetGridFieldValue routine and passes the requested field name.

3 Select Project | Compile rbFJProj. You should get an error relating to TppDataType in the GetGridFieldValue event handler.

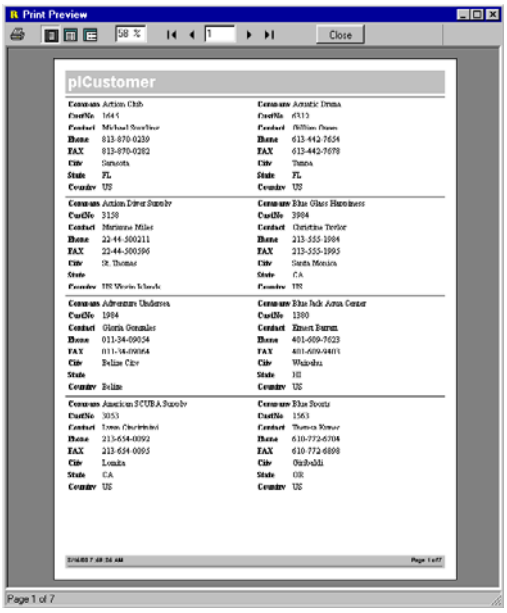
4 Add 'ppTypes' to the 'uses' clause at the top of the unit.

5 Compile and select File | Save from the Delphi main menu.

6 Select the report component and set the DeviceType to 'Screen'.

7 Run the project.

8 Click on the Preview button.



Using the Rich Text Component for Mail/Merge

Overview

This tutorial will show you how to do the following:

- Generate a form letter with address information from a database
- Edit the standard Rich Text component
- Use the mail/merge feature

Jennifer Davis
100 Cranberry St.
Wellesley, MA 02181

Dear Jennifer,
Greetings and welcome to the new standard for Delphi Reporting!

Over the past several months we have received many questions about Digital Metaphors and about the history behind ReportBuilder. When we first starting getting into Delphi, we were amazed at what could be accomplished using this slick object-oriented tool. It is a tribute to creators of Delphi that they were able to design something so powerful that it could actually be used to write itself! But then we hit the wall - there were no integrated tools available for building the type of professional quality reports to which we, and more importantly our *clients*, were accustomed. We started by taking the best features from the report designers we had worked with in the past. We then combined object-oriented design principles and our relentless pursuit of quality. The result is a product that has quickly become the report tool of choice among Delphi novices and experts alike.

Jennifer, we would like to invite you to download a demo of our software from www.digital-metaphors.com. Afterwards, drop us a line at info@digital-metaphors.com. We always love to hear from fellow Delphi enthusiasts.

Sincerely,

The Digital Metaphors Team

Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmMailMerge'.

3 Select File | Save As from the Delphi menu and save the form under the name rbMailM in the My RB Tutorials directory.

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbMMProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create a Table, DataSource, and DataPipeline Component

1 Select the Data Access tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblClient
TableName	clients.dbf

4 Add a DataSource component to the form.

5 Configure the DataSource component:

DataSet	tblClient
Name	dsClient

6 Select the RBuilder tab of the Delphi component palette.

7 Add a DBPipeline component  to the form.

8 Configure the DBPipeline component:

DataSource	dsClient
Name	plClient

Create a Report and Connect it to the Data

1 Add a Report component  to the form.

2 Configure the Report component:

DataPipeline	plClient
Name	rbMailMerge

Invoke the Report Designer and Set the Page Layout

1 Double-click on the Report component to display the Report Designer.

2 Size and move the Report Designer window so that the Object Inspector is visible.

3 Select File | Page Setup from the Report Designer main menu.

4 Select the Margins tab.

5 Set all margins to 1.25.

6 Click the OK button to close the dialog.

Modify the Bands

1 Select Report | Header from the Report Designer main menu. The header band will be removed from the report.

2 Select the Report | Footer option from the main menu. The footer band will be removed from the report.

3 Right-click over the white space of the detail band and select Position. Set the PrintCount to 1 and the Height to 4. This will allow only one detail band to print per page, thus creating the effect of one form letter per page.

4 Place a RichText component in the detail band.

5 Right-click over the RichText component and select the Stretch menu option. This will force the RichText component to resize based on the size of the letter.

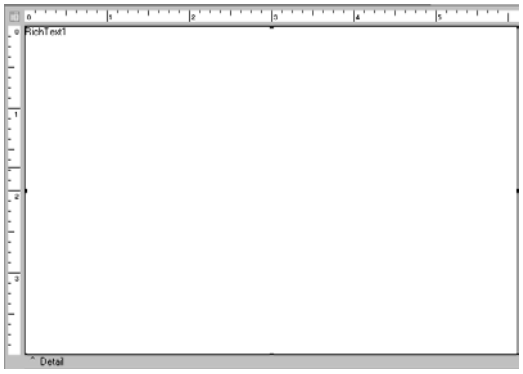
6 Right-click over the RichText component and set the position and size:

Left	0
Top	0
Width	6
Height	4

Note: We've positioned the RichText control so that it fills the entire detail band. This allows us to use the margins of the report to control the positioning of the letter, as opposed to positioning the RichText component within the band. The height of 4 is arbitrary: it will simply allow us to read the

entire contents of the letter while designing. When the report prints, the RichText component will calculate its height based on the length of the letter.

layout check



Load MMLetter into the RichText Component

1 Right-click over the RichText component and select the MailMerge menu option.

2 Right-click over the RichText component and access the Edit menu option. The Rich Text Editor will be displayed.

3 Select File | Open from the Rich Text Editor menu.

4 Locate the MMLetter.RTF file in the RBuilder\Tutorials directory. Open this file.

layout check

Add Fields to the Letter

1 Select Edit | Insert Field. A list of fields from the client table will be displayed.

2 Select the FIRST_NAME field and click OK.

3 Type a space and then insert the LAST_NAME field.

4 Press Enter to start a new line.

5 Insert the ADDRESS_1 line.

6 Press Enter to start a new line.

7 Insert the CITY field. Type a comma and a space.

8 Insert the STATE field.

9 Type a space and then insert the ZIP field.

10 Press enter twice to create a blank line below the address fields.

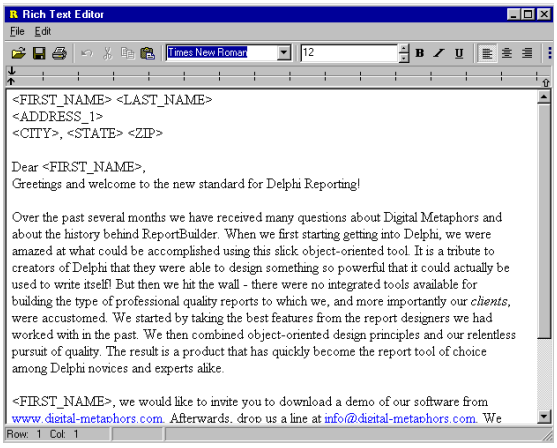
11 Type 'Dear', then a space, then insert the FIRST_NAME field.

12 Type a comma and press Enter twice.

13 Locate the start of the second paragraph (starts with the words 'We would like...').

14 Change this to '<FIRST_NAME>, we would like...'

layout check



15 Close the Rich Text Editor. Click the Yes button when the Save Changes dialog appears.

16 Select File | Save from the Delphi main menu.

17 Preview the report. The data from the clients table should be displayed.

Preview the Report at Run-Time

1 Close the Report Designer.

2 Select the Standard tab of the Delphi component palette.

3 Add a Button component to the form.

4 Configure the Button component:

Name	btnPreview
Caption	Preview

5 Add the following code to the OnClick event handler of the button:

```
rbMailMerge.Print;
```

6 Select File | Save from the Delphi main menu.

7 Select Project | Compile rbMMProj. Fix any compilation problems.

8 Run the Project.

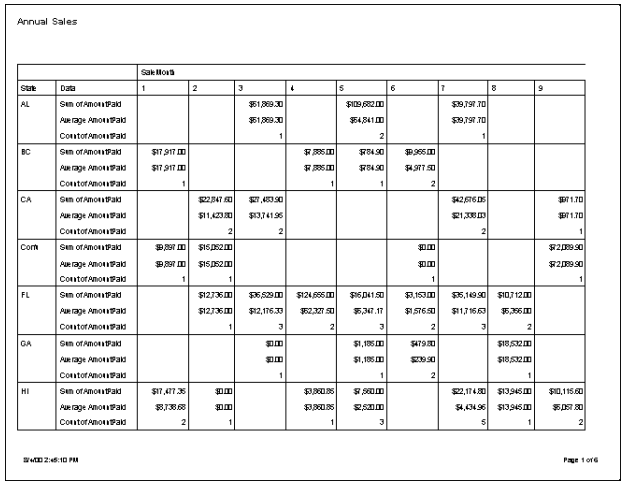
9 Click the Preview button. The report should be five pages, with one letter per page.

Creating a Crosstab

Overview

This tutorial will show you how to do the following:

- Summarize data in a grid format
- Create a crosstab report



Annual Sales

		Sale Month								
State	Data	1	2	3	4	5	6	7	8	9
AL	Sum of AmountPaid			$51,899.30		$129,682.00		$39,791.70		
	Average AmountPaid			$51,899.30		$64,841.00		$39,791.70		
	Count of AmountPaid			1		2		1		
BC	Sum of AmountPaid	$17,917.00			$7,895.00	$784.90	$9,955.00			
	Average AmountPaid	$17,917.00			$7,895.00	$784.90	$4,977.50			
	Count of AmountPaid	1			1	1	2			
CA	Sum of AmountPaid		$22,847.60	$27,483.90				$42,676.05		$971.70
	Average AmountPaid		$11,423.80	$13,741.95				$21,338.03		$971.70
	Count of AmountPaid		2	2				2		1
Conn	Sum of AmountPaid	$9,891.00	$15,052.00				$0.00			$72,089.90
	Average AmountPaid	$9,891.00	$15,052.00				$0.00			$72,089.90
	Count of AmountPaid	1	1				1			1
FL	Sum of AmountPaid		$12,736.00	$36,529.00	$124,655.00	$16,041.50	$3,153.00	$35,149.90	$10,712.00	
	Average AmountPaid		$12,736.00	$12,176.33	$62,327.50	$5,347.17	$1,576.50	$11,716.63	$5,356.00	
	Count of AmountPaid		1	3	2	3	2	3	2	
GA	Sum of AmountPaid			$0.00		$1,185.00	$419.00		$18,532.00	
	Average AmountPaid			$0.00		$1,185.00	$209.90		$18,532.00	
	Count of AmountPaid			1		1	2		1	
HI	Sum of AmountPaid	$17,477.35	$0.00		$3,860.85	$7,560.00		$22,174.80	$13,945.00	$10,115.60
	Average AmountPaid	$8,738.68	$0.00		$3,860.85	$2,520.00		$4,434.96	$13,945.00	$5,057.80
	Count of AmountPaid	2	1		1	3		5	1	2

9/4/00 2:46:10 PM Page 1 of 6

Create a New Application

Note: The crosstab component is available in the Professional and Enterprise editions only.

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmSaleCross'.

3 Select File | Save As from the Delphi menu and save the form under the name rbCross in the My RB Tutorials directory.

4 Select View | Project Manager from the main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbCrossProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create a Query, DataSource, and DataPipeline Component

1 Select the Data Access tab of the Delphi component palette.

2 Add a Query component to the form.

3 Configure the Query component:

DatabaseName	DBDEMOS
Name	qryOrder

SQL

```
SELECTAmountPaid, State,
EXTRACT (MONTH FROM SaleDate) AS
↳ SaleMonth
FROM Customer, Orders
WHERE Orders.CustNo = Customer.CustNo
```

4 Double-click on the Active property in the Object Inspector to set it to True.

5 Add a DataSource component to the form.

6 Configure the DataSource component:

DataSet	qryOrder
Name	dsOrder

7 Select the RBuilder tab of the Delphi component palette.

8 Add a DBPipeline component  to the form.

9 Configure the DBPipeline component:

DataSource	dsOrder
Name	plOrder

Create a Report

1 Add a Report component  to the form.

2 Configure the Report component:

Name	rbOrder

3 Double-click on the Report component to display the Report Designer.

4 Position the Report Designer so that the Object Inspector is also visible.

5 Place a crosstab component  (from the Advanced toolbar) in the detail band.

6 Use the Edit toolbar to assign the crosstab to the plOrder data pipeline.

7 Select File | Page Setup. Access the Paper Size and set the orientation to Landscape. Click OK.

8 Right-click over the crosstab and select Configure. The Crosstab Designer will be displayed.

9 Select the SaleMonth field (at the bottom of the list) and drag it over the new row cell. Black, triangular indicators should show where the new cell will be created.

10 When these indicators appear to the left of the new row cell, release the mouse button.

11 Click OK.

12 Select File | Save from the Dephi main menu.

13 Preview. A blank page is displayed because no Value dimensions have been created for the crosstab.

Design the Crosstab

1 Return to the design workspace, right-click over the crosstab, and select Configure.

2 Drag the Amount Paid field over the new value cell. When the indicators appear, release the mouse button.

diagram check

		new column
SaleMonth	new row	Sum of AmountPaid 1000
		new value
Grand Total		1000

Note: The number 1000 represents the format of the calculated value. The Grand Total indicates that the last row of the crosstab will show the total Amount Paid for all months.

Add Additional Values to the Crosstab

1 Drag the Amount Paid field over the new value cell and release it.

2 Select the second Sum of Amount Paid value.

3 Locate the drop-down list box on the toolbar.

4 Select Average from the drop-down list.

5 Once again, drag the Amount Paid field over the new value cell and release it.

6 Select the second Sum of Amount Paid (the one below the average).

7 Select Count from the drop-down list.

diagram check

		new column
SaleMonth	new row	Sum of AmountPaid 1000
		Average AmountPaid 1000
		Count of AmountPaid 1000
		new value
Grand Total (Sum of AmountPaid)		1000
Grand Total (Average AmountPaid)		1000
Grand Total (Count of AmountPaid)		1000

8 Click OK.

9 Select File | Save from the Delphi main menu.

10 Preview. The crosstab includes new values. Advance to the second page. The grand totals are on this page. The report tells us the sum, average, and count for the Amount Paid per month.

Set the Format of the Values

1 Access the Crosstab Designer.

2 Select the 1000 under Sum of Amount Paid.

3 Right-click and select Display Format. Select the first menu option with a dollar sign.

4 Select the 1000 under Average Amount Paid.

5 Right-click and select Display Format. Select the first menu option with a dollar sign.

Calculate Totals by State

1 Move the SaleMonth cell over the new column cell and release.

2 Drag State from the field list, place it over the new row cell, and release.

diagram check

		SaleMonth	Grand Total
		new column	
State	new row	Sum of AmountPaid $1,000.00	$1,000.00
		Average AmountPaid $1,000.00	$1,000.00
		Count of AmountPaid 1000	1000
		new value	
Grand Total (Sum of AmountPaid)		$1,000.00	
Grand Total (Average AmountPaid)		$1,000.00	
Grand Total (Count of AmountPaid)		1000	

3 Click OK.

4 Select File | Save from the Delphi main menu.

5 Preview. Select Whole Page on the preview toolbar. The crosstab shows us the sum, average, and count of the Amount Paid per month for each state. Notice that the months go across the top of the report instead of down the side. This is because SaleMonth is now a column dimension.

Lay Out the Header Band

1 Return to the design workspace and place a label in the upper left corner of the header band.

2 Set the caption to Monthly Sales by State.

3 Configure the label:

Font Name	Arial
Font Size	12
Font Style	Bold
Font Color	Black

4 Place a SystemVariable in the lower left corner of the footer band.

5 Set it to PrintDateTime.

6 Place another SystemVariable in the lower right corner of the footer band.

7 Set it to PageSetDesc.

8 Align the top of the SystemVariables.

9 Configure the SystemVariables:

Font Name	Arial
Font Size	8
Font Style	Regular
Font Color	Black

10 Select File | Save from the Delphi main menu.

11 Preview. Notice that the page number prints on each page.

Set Pagination

1 Right-click over the crosstab component and select Pagination. Notice that the default setting is Down then Across.

2 Select Across then Down.

3 Preview. The pages print across then down.

Use Repeated Captions

1 Right-click over the crosstab component.

2 Expand the Style menu option. Notice that the default setting is Standard.

3 Select Repeated Captions.

4 Preview and print all four pages of the crosstab. Notice how the repeating captions clarify the context of the crosstab.

RAP TUTORIALS

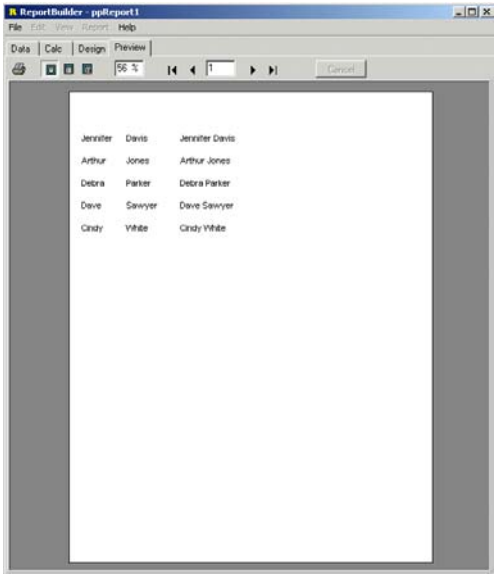
RAP TUTORIALS

Concatenating Fields

Overview

This tutorial will show you how to do the following:

- Create a basic report at design-time
- Create a report that concatenates two fields using RAP instead of Delphi event handlers.



Create a Tutorial Folder

1 Access the Windows Explorer.

2 Create a folder on the root of your hard drive.

3 Name the folder My RAP Tutorials.

Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a new form.

2 From the RBuilder tab on the Component Palette, select a Report component and drop it on the blank form.

3 Right-click the component and select Report Designer from the context menu.

Create a DataView

1 In the open Report Designer, click the Data tab.

2 Select File | New from the menu. The New Items dialog will appear.

3 Double-click the Query Wizard to begin creating a dataview. From the list of available tables, double click the Clients table.

4 Select Next on all of the following pages until you reach the last page, then click Finish.

Layout the Report

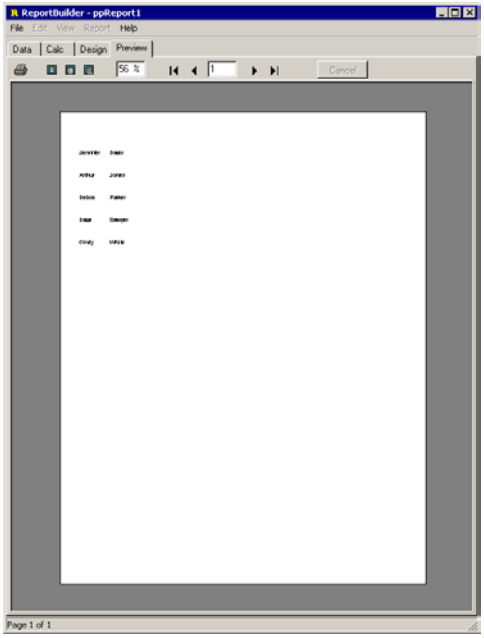
1 Click the Design tab and turn off the report's header and footer bands by selecting those items in the Report menu.

2 Add two DBText controls to the Detail band.

3 Click on DBText1 and use the Data Field drop down list to select First Name.

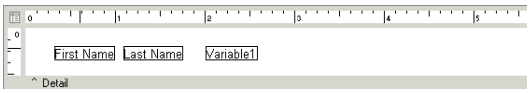
layout check

4 Click on DBText2 and use the Data Field drop down list to select Last Name. If you preview the report now, you should see five detail records with first and last names displayed.



5 Place one Variable component to the right of the DBText components in the detail band. The Variable will display the concatenated fields.

layout check



6 Select all of the components by shift-clicking, and select the Align Top icon on the Align or Space Toolbar.

Navigate the Calc Workspace

We will use the OnCalc event of the Variable to concatenate the two fields.

1 Click the Calc tab to display the Calc Workspace.

2 Right-click the Code Explorer's tree view and select Events.

3 Click Variable1.

4 Right-click the OnCalc event and select New.

Add the Concatenation Code

The Data tab of the Code Toolbox should be active, if it is not, select it. In the upper window of the Code Toolbox, you should see the pipeline we have added, plClients. Below that, you should see an entry for each field in the pipeline. These items are draggable.

We're going to enter the following line of code, but we are going to construct it via drag and drop.

1 Click on the First Name entry in the Toolbox and drag it to the Code Editor, just to the right of the "Value := " line.

2 Drag the Last Name entry to the right of the line of code.

3 Type in the remaining characters of the line as shown below.

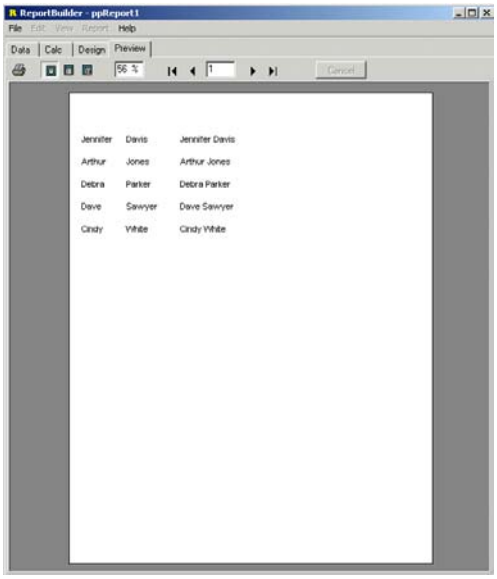
Code	Variable1 OnCalc Event

```
procedure Variable1OnCalc(var Value: Variant);
begin

  Value := plClients['First Name'] + ' ' + plClients['Last Name'];

end;
```

Compile and Preview

1 To compile your code, right-click on the Code Editor and select Compile.

2 To view the results, click the Preview tab.



Congratulations--you've successfully concatenated database fields using RAP!

Save the Report

1 Click on the Design tab and select Save As from the File menu.

2 Navigate to the My RAP Tutorials folder and save the report as RAP Tutor 1.rtm.

Note that the RAP Code, the DataView, and the report layout have all been saved together.

Load the Report

1 Select New Report  from the File menu.

2 Go back to the File menu and open the report you just saved.

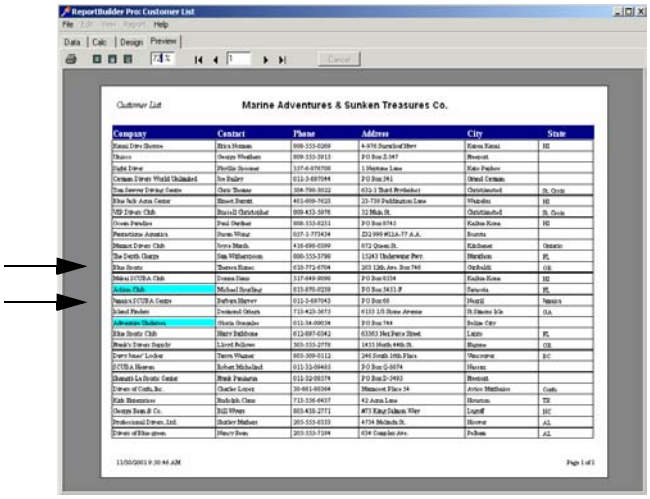
If you preview the report, you will see that the RAP code is executing and you did not have to reconnect any event handlers.

Color-Coding Components

Overview

This tutorial will show you how to do the following:

- Use RAP at run-time
- Add local variables
- Add event handler code to color-code a component.



Open the Demo Project

1 Select File | Open | RBuilder | Demos | 3. EndUser | 1. Report Explorer | EndUser.dpr

2 Enable RAP by removing the "x" from in front of $DEFINE RAP in the MyEurpt.pas interface.

3 Compile and run the demo project.

4 Select the Customers folder.

5 Double-click the Customer List to open it.

Enable DetailBeforePrint

1 Click the Calc tab to activate the Calc workspace.

2 Right-click the left-hand pane tree view in the Code Explorer and select Events.

3 Scroll through the Report Tree until you find the Detail band and select it.

4 Right-click the BeforePrint event in the right-hand pane and select New.

Add Local Variables

1 Just as you would in Delphi, place your cursor just before the begin and add the local variables.

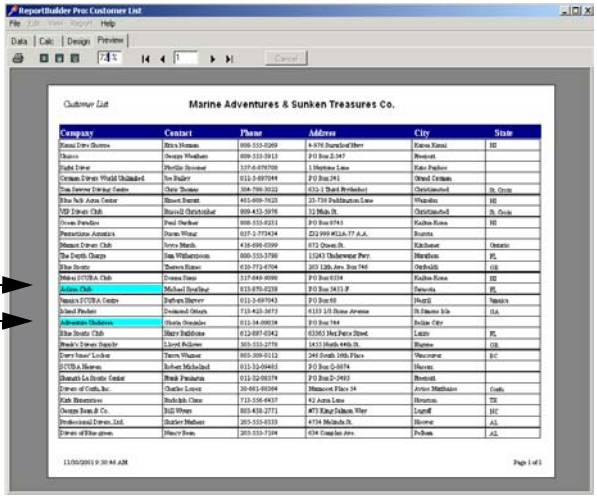
2 Place your cursor between the begin and end and enter the code for the event handler as shown below.

Code	Detail BeforePrint Event

```
procedure DetailBeforePrint;
var
 lsString: String;
 liLength: Integer;

begin

 lsString := plCustomer['Company'];
  liLength := Length(lsString);
  Delete(lsString, 2, liLength -1);

  if lsString = 'A' then
   ppCustomerListDBText2.Color := clAqua

  else
   ppCustomerListDBText2.Color := clWhite;

end;
```

Preview and Save

1 Click the Preview tab to view the report. You should see some of the company names highlighted in aqua.



It is possible to save your code as a separate entity from your report by exporting it. You can also import saved code.

1 Click the Calc tab and select the File menu.

2 Select Export, and give your code a name.

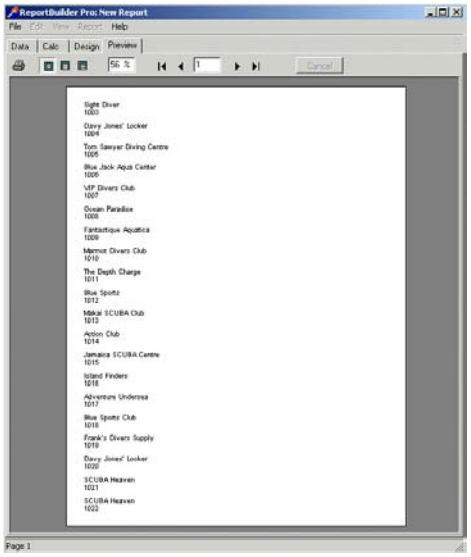
3 Close the Report Designer. You should see your code in the folder where you saved it.

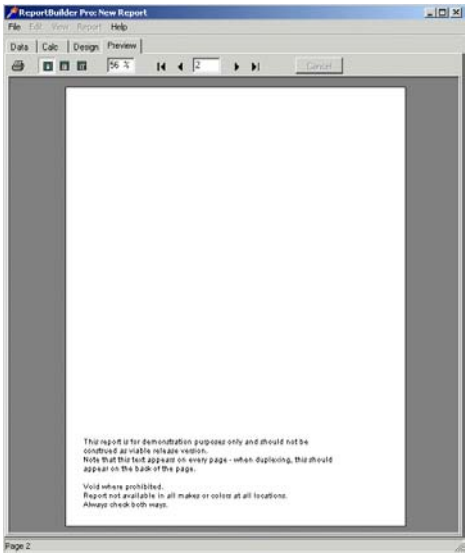
Dynamic Duplexing

Overview

This tutorial will show you how to do the following:

- Use duplexing to print a disclaimer on the back of every page of a report.
- Print the data only on even numbered pages and the disclaimer on the odd numbered pages.





Enable RAP

1 Select File | Open | RBuilder | Demos | 3. EndUser | 1. Report Explorer | EndUser.dpr.

2 Enable RAP by removing the "x" in front of $DEFINE in the MyEUrpt.pas interface.

3 Build and run the demo project.

4 Navigate to the folder of your choice and click the New Report button. The Report Designer will open automatically.

Create a New DataView

1 Click the Data tab.

2 Select File | New, and choose the Query Wizard from the New Items dialog box.

3 On the first page of the Wizard add Customer to the Selected Tables list.

4 Add Orders to the Selected Tables list. This should cause the Join Table dialog to appear.

5 Click on the TaxRate row in the Joined Fields list box and click the Remove button.

6 Click OK to join the tables.

7 Accept the defaults on all the rest of the pages and click Finish to create the DataView.

8 Close the preview when it appears.

Create Report Layout

1 Click the Design tab to begin laying out the report.

2 Turn off the Header and Footer bands by selecting those items in the Report Menu.

3 Select Report | Groups to display the Groups dialog.

4 Select Customer.CustNo from the Groups drop down list and click Add to create the group.

5 Make sure the Reprint Group Headers on Subsequent Pages option is checked and click OK to create the group.

6 Right-click the band divider labelled ^Group Header[0]: CustNo and select Position from the context menu.

7 We need to make the Group Header take up the entire physical page, so assuming that Report.Units is set to Inches and your margins are set to .25" each, set Height to 10.5" and click OK.

8 Place a Memo in the Group Header band to contain your disclaimer.

9 Right-click the memo and select Lines.

10 Edit the memo control and add some official sounding text.

Note: The group header and memo control will be printing on the back of every page.

11 Next, place two DBText controls in the detail band and attach one to the Company field and one to the OrderNo field.

Note: The group header will work for every page but the last one. Since the data will always end on an odd numbered page, we'll need to arrange something to print the disclaimer on the back of the last page.

12 Position the Company DBText component:

Left	0.0313
Top	0.0208

13 Position the OrderNo DBText component:

Left	0.0313
Top	0.2083

14 Right-click the DBText components and select AutoSize.

15 Select the Summary item from the Report menu to turn on the Summary band.

16 Just as you did for the group header band, set the height of the summary band to 10.5".

17 Copy the Memo control from the group header to the summary band and place it in the same position in the band.

18 Click the Calc tab.

19 Right-click on the Code Explorer's treeview and select Events.

20 Right-click on the Report's OnStartPage event in the listview and select New.

21 In the Code Editor, enter the code below:

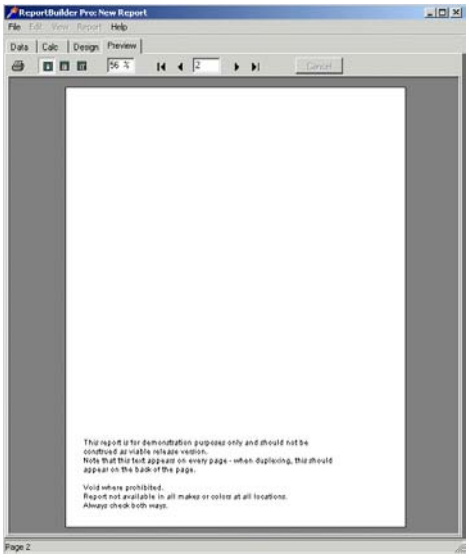
Code	Report OnStartPage Event

```
procedure ReportOnStartPage;
begin

  GroupHeaderBand1.Visible := (Report.AbsolutePageNo mod 2=0);

end;
```

Compile and Preview

1 To compile your code, right-click on the Code Editor and select Compile.

To view the results, click the Preview tab



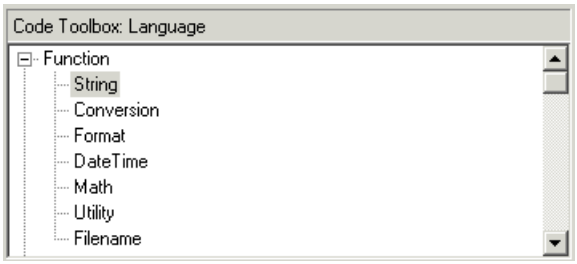
As you scroll through the report you'll notice that every other page contains the disclaimer. The summary band should make sure that the last page of the report is also the disclaimer. If you print this report in duplex mode, the disclaimer should appear on the back of every page.

Adding Functions to RAP

Overview

This tutorial will show you how to do the following:

- Add two functions and a category to the functions list in the Code Toolbox.
- Add a pass-through function to retrieve the application's filename, a function to expose the ExtractFilename Delphi function, and a category named "Filename."



Create a New Unit

1 Open Delphi and select Close All from the file menu.

2 Select File | New | Unit from the Delphi menu.

3 Select File | Save As | c:\ My RAP Tutorials

4 Save the file as myRapFuncs.pas. This unit will contain our new pass-through functions. This unit will contain our new pass-through functions.

Edit Uses Clause

1 Add a uses clause between the interface and implementation sections.

2 Add Forms, raFunc, and ppRTTI to the clause. The code should look like this:

```
unit myRapFuncs;

interface

uses
     Forms, raFunc, ppRTTI;

implementation
```

Note: raFunc contains TraSystemFunction (from which all pass-through functions descend) and the raRegisterFunction, which is used to register your pass-through functions with RAP.

Add TraSystemFunction Descendent for Category

1 We need a new TraSystemFunction descendant to define a new category. Add a type section under the uses clause of your new unit. We will add a descendant to override TraSystemFunction.Category. Add the following class declaration to the type section:

Code	TmyFilenameFunction Class Declaration

```
type

  TmyFilenameFunction = class(TraSystemFunction)
  public
    class function Category: String; override;
  end;
```

2 In the unit's implementation section, add the following:

Code	Category Method

```
implementation

class function TmyFilenameFunction.Category: String;
begin

  result := 'Filename';
end;
```

Now anything that inherits from TmyFilenameFunction will appear in the Filename category in the Code Toolbox.

This class will not be registered with RAP since it does not actually implement a pass-through function.

Add New Function Class

Under the TmyFilenameFunction class declaration, add the following class declaration:

Code	TmyApplicationFilenameFunction Class Declaration

```
TmyApplicationFilenameFunction = class (TmyFilenameFunction)
  public
    procedure ExecuteFunction(aParams: TraParamList); override;
    class function GetSignature: String; override;
    class function HasParams: Boolean; override;

  end;
```

Implement ExecuteFunction

In the unit's implementation section, enter the following:

Code	ExecuteFunction Method

```
procedure TmyApplicationFilenameFunction.ExecuteFunction(aParams: TraParamList);
var
  lsResult: String;

begin

  lsResult := Application.Exename;
  SetParamValue(0, lsResult);

end;
```

Implement GetSignature

In the unit's implementation section, enter the following:

Code	GetSignature Method

```
class function TmyApplicationFilenameFunction.GetSignature: String;
begin

  Result := 'function ApplicationFilename: string;';

end;
```

Implement HasParams

In the unit's implementation section, enter the following:

Code	HasParams Method

```
class function TmyApplicationFilenameFunction.HasParams: Boolean;
begin

  result := False;

end;
```

Add Initialization

1 At the bottom of the unit, before the final end, add an initialization section.

2 In this section, enter the following:

Code	ApplicationFilename Function Registration

```
initialization

  raRegisterFunction('ApplicationFilename',TmyApplicationFilenameFunction);

end.
```

This line actually registers the pass-through function with RAP.

Add myRapFuncs to a Project

Now we need to see your new function in action. We will add the unit to one of the End User demos.

1 Open the RBuilder | Demos| 3. EndUser | 1. Report Explorer | EndUser.dpr project.

2 Open the main form, myEURpt.pas.

3 Scroll down until you see a series of *{$DEFINE}* statements. Look for the one that defines RAP and remove the 'x' from in front of the *$DEFINE*.

Now, we need to add myRapFuncs to the project.

4 Select Project | Add to Project.

5 Navigate to My RAP Tutorials and select myRapFuncs.pas.

6 Go to the bottom of the Uses clause and add myRapFuncs to the clause.

7 Compile and run the project.

8 When the main form appears, click Launch.

9 Navigate to the Customers folder in the Report Explorer.

10 Double-click the Customer List report.

Note: You should see a Calc tab displayed in the Report Designer. If you do not, go back and make sure you enabled RAP and select Build All in Delphi.

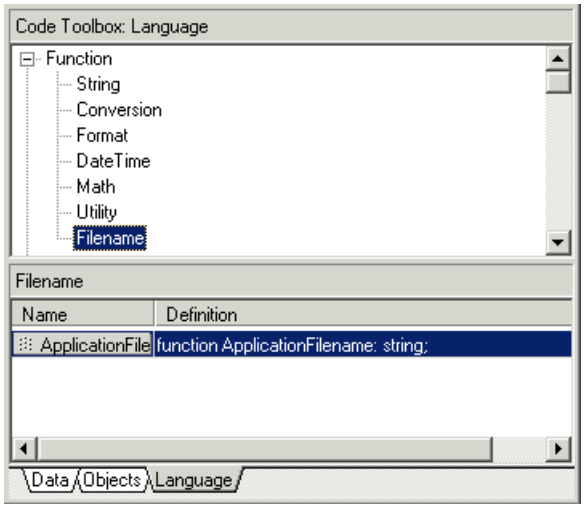
See ApplicationFilename in Code Toolbox

1 Click the Calc tab.

2 Click the Language tab in the Code Toolbox and scroll to the top of the Toolbox treeview to find the Function node.

3 Click the Filename node.

In the function list you should see the fruits of your efforts.



Now let's see it in action...

Add ApplicationFilename to Code

1 Right-click on the Report Explorer's tree view and select Events.

2 Select the Report node, if it is not already selected, and click the BeforePrint event listed in the right-hand pane.

3 Click the Code Editor to activate the event handler stub and add a local variable as shown below.

4 Add the code for the ReportBeforePrint event.

5 Right-click on the Code Editor and select Compile.

6 Assuming the code compiles, click the Preview tab.

Congratulations! You should see a message dialog with the application's filename.

Code	Report BeforePrint Event

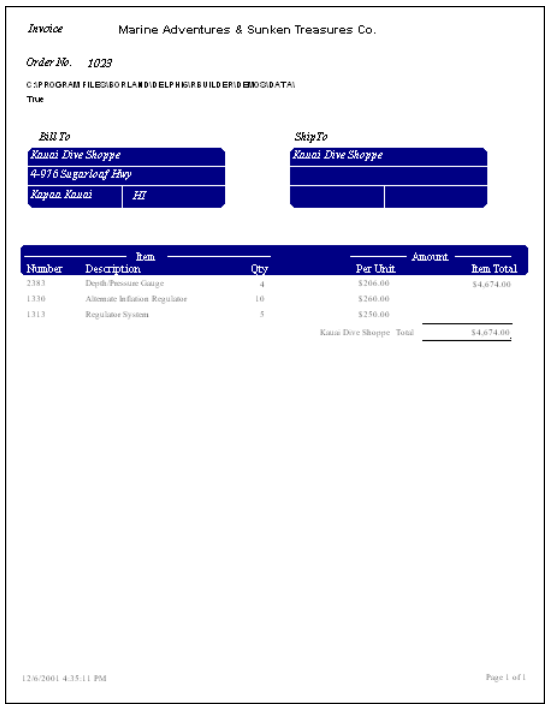
```
var
 lString: String;
begin

  lString := ApplicationFilename;
  ShowMessage(lString);

end;
```

Extending the RAP RTTI

Overview

This tutorial will show you how to do the following:

- Make RAP aware of a new component.
- Register a new class to surface TDataBase within RAP.
- Add support for the public property, Directory and the public method, ValidateName.



Create a New Unit

1 Select File | Open | RBuilder | Demos | 3. EndUser | 1. Report Explorer | EndUser.dpr.

2 Select File | New | Unit, to create a new unit.

3 Save the unit as myRapClass.pas in the C: | My RAP Tutorials directory. This unit will contain our new TraRTTI descendant.

4 In the new unit, add a uses clause between the interface and implementation sections and add Classes, Forms, raFunc, ppRTTI, db, dbTables and ppUtils to the clause.

```
unit myRapClass;

interface

uses
Classes, Forms, raFunc, ppRTTI, db, dbTa-
bles, ppUtils
```

Declare TmyTDataBaseRTTI

Next, we must find the proper TraRTTI descendant from which to descend this class. TDataBase descends from TCustomConnection which descends from TComponent. There is no TraRTTI descendant for TCustomConnection, but there is one for TComponent (TraTComponentRTTI) so we'll use that one.

1 Add a type section below the Uses clause to contain our new TmyTDataBaseRTTI component.

2 Add the following declaration to the type section:

Code	TmyTDataBaseRTTI Class Declaration

```
type

TmyTDataBaseRTTI = class(TraTComponentRTTI)
    public
      class procedure GetPropList(aClass: TClass; aPropList: TraPropList); override;
      class function  RefClass: TClass; override;
      class function  GetPropRec(aClass: TClass; const aPropName: String; var aPropRec:
                      TraPropRec): Boolean; override;
      class function  CallMethod(aObject: TObject; const aMethodName: String; aParams:
                      TraParamList; aGet: Boolean): Boolean; override;
      class function  GetParams(const aMethodName: String): TraParamList; override;
      class function  GetPropValue(aObject: TObject; const aPropName: String; var
                      aValue): Boolean; override;
      class function  SetPropValue(aObject: TObject; const aPropName: String; var
                      aValue): Boolean; override;

  end;
```

Implement GetPropList

Implement the overridden method, GetPropList with the following code:

Code	GetPropList Method

```
class procedure TmyTDataBaseRTTI.GetPropList(aClass: TClass; aPropList: TraPropList);
begin

  inherited GetPropList(aClass, aPropList);

  aPropList.AddProp('Directory');
  aPropList.AddMethod('ValidateName');

end;
```

Implement RefClass

Add the following code to the implementation section:

Code	RefClass Method

```
class function TmyTDataBaseRTTI.RefClass: TClass;
begin

  Result := TDataBase;

end;
```

Implement GetPropRec

Implement the overridden method, GetPropList with the following code:

Code	GetPropRec Method

```
class function TmyTDataBaseRTTI.GetPropRec(aClass: TClass;
               const aPropName: String; var aPropRec: TraPropRec): Boolean;
begin

  Result := True;
  if ppEqual(aPropName, 'Directory') then
    PropToRec(aPropName, daString, False, aPropRec)
  else if ppEqual(aPropName, 'ValidateName') then
    MethodToRec(aPropName, True, aPropRec)
  else
    Result := inherited GetPropRec(aClass, aPropName, aPropRec);

end;
```

Implement CallMethod

Implement the overridden method, CallMethod,
with the following code:

Code | CallMethod Method

```
class function TmyTDataBaseRTTI.CallMethod(aObject: TObject; const aMethodName: String;
               aParams: TraParamList; aGet: Boolean): Boolean;
var
  lsName: String;
  lbResult: Boolean;
  lDataBase: TDataBase;
begin

  Result := True;
  lDataBase := TDataBase(aObject);
  if ppEqual(aMethodName, 'ValidateName') then
    begin
      aParams.GetParamValue(0, lsName);
      try
        lDataBase.ValidateName(lsName);
        lbResult := True;
      except
        On EDatabaseError do
          lbResult := False;
      end;
      aParams.SetParamValue(1, lbResult);
    end
  else
    Result := inherited CallMethod(aObject, aMethodName, aParams, aGet);

end;
```

Implement GetParams

Implement the overridden method, GetParams,
with the following code:

Code | GetParams Method

```
class function TmyTDataBaseRTTI.GetParams(const aMethodName: String): TraParamList;
begin

  if ppEqual(aMethodName, 'ValidateName') then
    begin
      Result := TraParamList.Create;
      Result.AddParam('Name', daString, nil, '', True, False);
      Result.AddParam('Result', daBoolean, nil, '', False, False);
    end
  else
    Result := inherited GetParams(aMethodName);

end;
```

Implement GetPropValue

Implement the overridden method, GetPropValue,
with the following code:

Code	GetPropValue Method

```
class function TmyTDataBaseRTTI.GetPropValue(aObject: TObject;
               const aPropName: String; var aValue): Boolean;
begin

  Result := True;
  if ppEqual(aPropName, 'Directory') then
    String(aValue) := TDataBase(aObject).Directory

  else
    Result := inherited GetPropValue(aObject, aPropName, aValue);

end;
```

Implement SetPropValue

Implement the overridden method, SetPropValue,
with the following code:

Code	SetPropValue Method

```
class function TmyTDataBaseRTTI.SetPropValue(aObject: TObject;
               const aPropName: String; var aValue): Boolean;
begin

  Result := True;
  if ppEqual(aPropName, 'Directory') then
    TDataBase(aObject).Directory := String(aValue)

  else
    Result := inherited SetPropValue(aObject, aPropName, aValue);

end;
```

Register the Class

In the unit's initialization section, add the following
call:

```
raRegisterRTTI(TmyTDataBaseRTTI);
```

In the finalization section, add the following call:

```
raUnregisterRTTI(TmyTDataBaseRTTI);
```

Create Access Function

For our purposes, we need to be able to get to a TDataBase so we can test the new RTTI class. We could simply create one, but let's get one thatal-ready exists. To do this, we'll create a new pass-through function that will allow us to get to a Del-phi object by name. Creating pass-through func-tions is covered in another tutorial, so we'll speed through this.

1 Add a new TraSystemFunction descendant, TmyDevelopersFunction.

Code	TmyDevelopersFunction

```
type

TmyDevelopersFunction = class (TraSystemFunction)
public
  class function Category: String; override;

end;


implementation

class function TmyDevelopersFunction.Category: String;
begin

  Result := 'Developer';

end;
```

2 Add a new TmyDevelopersFunction, TmyGet-DelphiComponentFunction.

Code	TmyGetDelphiComponentFunction

```
TmyGetDelphiComponentFunction = class (TmyDevelopersFunction)
public
  procedure ExecuteFunction(aParams: TraParamList); override;
  class function GetSignature: String; override;
end;

implementation

procedure TmyGetDelphiComponentFunction.ExecuteFunction(aParams: TraParamList);
var
  lsString: String;
  lResult: TComponent;
begin

  GetParamValue(0, lsString);
  lResult := Application.MainForm.FindComponent(lsString);
  SetParamValue(1, lResult);
end;

class function TmyGetDelphiComponentFunction.GetSignature: String;
begin
  Result := 'function GetDelphiComponent(const aComponentName: String): TComponent;';

end;
```

3 To register the new pass-through function, add the following to the initialization section

```
raRegisterFunction('GetDelphiComponent', TmyGetDelphiComponentFunction);
```

Add myRapClass to a Project

We will add the unit to one of the End User demos.

1 Open the RBuilder | Demos | 3. EndUser | 1. Report Explorer | EndUser.dpr project.

2 Open the main form, myEURpt.pas.

3 Scroll down until you see a series of {$DEFINE} statements. Look for the one that defines RAP and remove the 'x' from in front of the $DEFINE.

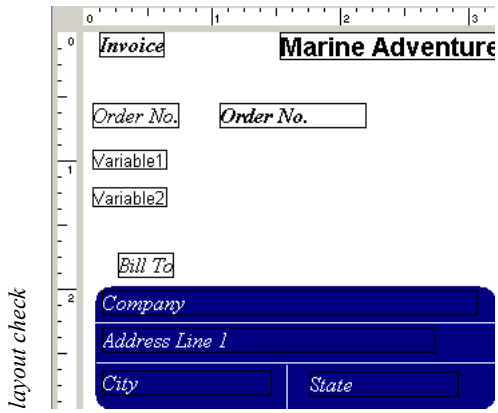
4 Go to the bottom of the uses clause and add myRapClass to the clause.

5 Compile and run the project.

6 Navigate to the Invoices folder in the Report Explorer.

7 Double-click the Invoices report.

8 In the Design tab, add two Variables to the Header band, one below the other. We'll use these variables to display information from our TData-Base.



Write RAP Code

1 Click the Calc tab.

2 Right-click the Code Explorer treeview and select Module.

3 Select the Declarations node, right-click the Variables item and select New.

4 Enter the following into the Code Editor:

```
gDataBase: TDataBase;
gbValidName: Boolean;
```

5 Now select the Events node, right-click OnCreate and select New.

6 In the event handler, enter the code for the Global OnCreate event.

7 Right-click the Code Explorer treeview and select Events.

8 Right-click the Code Explorer treeview and select Events.

9 Select Variable1 in the treeview.

10 Right-click the OnCalc event and select New.

11 Enter the following code in the event handler:

```
Value := gDataBase.Directory;
```

12 Select Variable2 in the treeview.

13 Right-click the OnCalc event and select New.

14 Enter the following code in the event handler:

```
if gbValidName then
  Value := 'True'
else
  Value := 'False';
```

Code	Global OnCreate Event

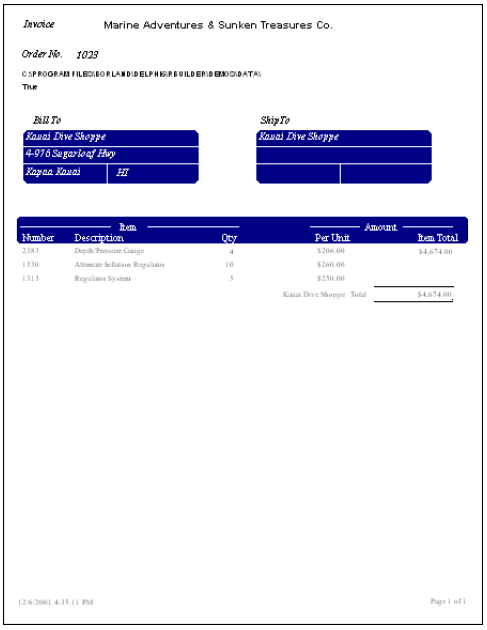
```
gDataBase := TDataBase(GetDelphiComponent('euDatabase'));
if (gDataBase <> nil) then
  begin

    gbValidName := gDataBase.ValidateName('DBDEMOS');

  end;
```

Preview the Report

Now we will preview the report. Note that when we call ValidateName, an exception will be raised and the program's execution will halt in Delphi's debugger. However, keep in mind that if you were running this application outside of Delphi, the exception would be handled and the user would see no problem.

1 Click the Preview tab.

You should see your two variables, one indicating a path and the other a boolean value.



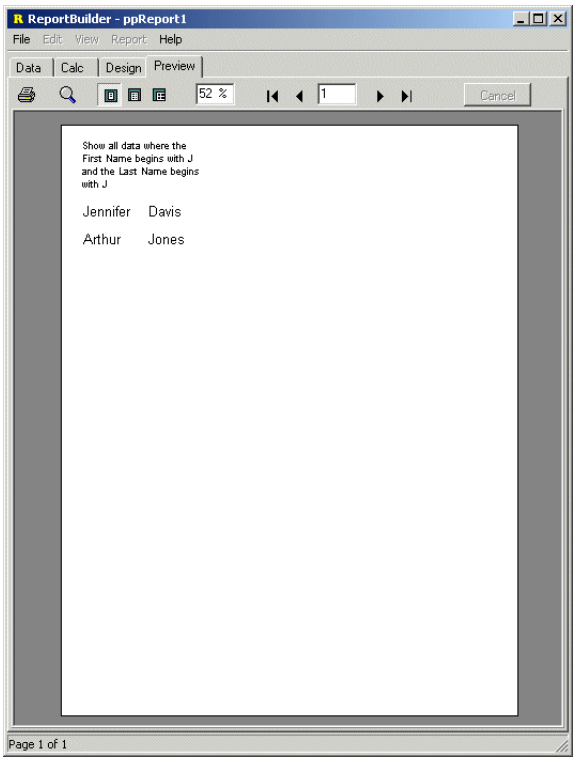
Congratulations! You have successfully added your first class to RAP.

Printing a Description of AutoSearch Criteria

Overview

This tutorial will walk you through the following

- Gain access to the AutoSearch field descriptions via RAP
- Create a new report and dataview for this tutorial.



Create a New Report

1 Open Delphi and select Close All from the File menu.

2 Select File | New Project.

3 Place a Report component on the new form.

Note: RAP can be used in Delphi design-time or at run-time. Obviously your users will use the Calc Workspace at run-time but you can use it to create your reports in either environment. This tutorial will be done in design-time.

4 Open the ReportDesigner by double-clicking the Report component.

5 Click the Data tab.

6 Select File | New.

7 Select Query Designer from the New Items dialog and click OK.

8 On the Tables tab, double-click Clients.

9 On the Fields tab, check the All Fields checkbox.

Next we will create search criteria...

Create Search Criteria

1 On the Search tab, double-click First Name to add a criteria.

2 For the new criteria, set Operator to "Like" and set Value to "J."

3 Check the AutoSearch checkbox.

4 Double-click Last Name to add a criteria.

5 For this criteria, set Operator to "Like" and set Value to "J."

6 Check the AutoSearch checkbox.

7 Right-click the second criteria and select "Insert OR".

8 Click the OK button in the Query Designer. You should now see the DataView.

9 Click the Preview button on the DataView. You should see two records displayed, Jennifer Davis and Arthur Jones.

Layout the Report

1 Click the Design tab.

2 Turn off the Footer band by selecting Footer from the Report menu.

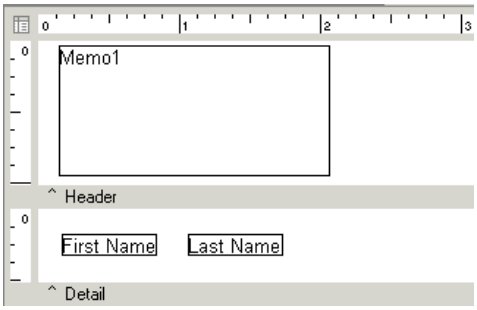
3 Drop two DBText controls onto the Detail band.

4 Assign DBText1 to First Name and DBText2 to Last Name.

5 Drop a Memo in the Header band.

6 Set the Memo to Stretch – this will hold the AutoSearch field descriptions.

layout check



Write the Code

1 Click the Calc tab.

2 Right-click the Code Explorer's treeview and select Events.

3 Click the Report node.

4 Right-click the OnStartPage event in the list-view and select New.

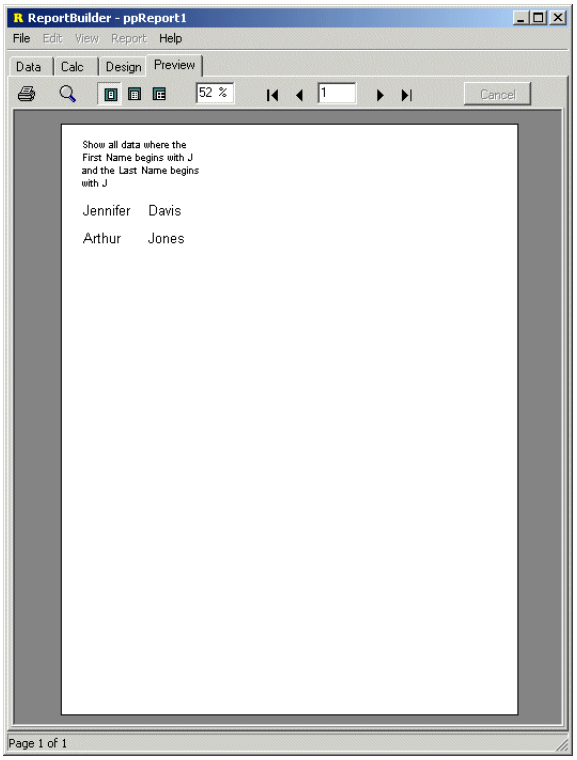
5 In the event handler stub in the Code Editor, enter the following code (note that you can either type this in or drag and drop the code from the Code Toolbox):

```
Report.GetAutoSearchDescription-
Lines(Memo1.Lines);
```

Compile and Preview

1 To compile your code, right-click on the Code Editor and select Compile.

2 To view the results, click the Preview tab.



Congratulations! You should see a description of the AutoSearch fields in the Memo.

Displaying Delphi Forms from RAP

Overview

The AutoSearch functionality is well suited to polling the end user for search criteria, however sometimes it is necessary to ask the end user for information that is not related to the database. In Delphi this would be easy, but the end user, without Delphi, would be unable to show custom forms unless you expose them via pass-through functions.

You will learn the following from this tutorial:

- The process of making custom forms available to the end use in RAP.
- Create custom forms and then make them available from within RAP.
- Create a report which accesses these forms.

Create User Input Form

1 Open Delphi and Select File | Close All.

2 Select File | New Form.

3 Configure the form's properties:

Name	frmMyUserInput
Height	211
Width	387
BorderStyle	bsDialog
Position	poScreenCenter
Caption	'User Input'

4 Save the new unit in the RBuilder | Demos | 3. EndUser | 1. Report Explorer directory as myUserInputForm.pas.

Add Components

1 Add a PageControl component to the form and set its properties:

Name	pgCtlMain
Height	141
Align	alTop
Style	tsFlatButtons

2 Place a Button on the form and set its properties:

Name	bnOK
Left	110
Top	148
Caption	'OK'
Default	True
ModalResult	mrOK

3 Place another Button on the form and set its properties:

Name	bnCancel
Left	194
Top	148
Caption	'Cancel'
ModalResult	mrCancel

4 With the existing PageControl, add two new TabSheets. Set the names:

Name = tabStartup
Name = tabMasterRecord

Note: The TabVisible property for both these tabsheets will be set to False in the form's OnCreate event

5 On tabStartup add a Label and set its properties:

Name	lblHeading
Left	8
Top	16
Width	43
Caption	'Heading:'

6 Place an Edit on tabStartup and set its properties:

Name	ebHeading
Left	68
Top	12
Width	281
Text	'Enter a Report Heading'

7 Now, place a RadioGroup on tabStartup and set its properties:

Name	rgDestination
Left	8
Top	44
Width	345
Height	61
Caption	'Destination '
Columns	2
ItemIndex	0
Items.Strings	'Marketing'
	'CEO'
	'Vice Presidents'
	'Files'

8 On tabMasterRecord add 2 Labels and set their properties:

Name	lblCompany
Left	12
Top	8
Width	47
Caption	'Company:'

Name	lblCompanyName
Left	68
Top	8
Width	82
Caption	'CompanyName'

9 Next, add a CheckBox to tabMasterRecord, and set its properties:

Name	cbIncludeAddress
Left	32
Top	40
Width	181
Caption	'Include Address Info in Report'

Code Form Methods

1 Next, add an OnCreate method for the form and enter the following code:

```
tabStartup.TabVisible := False;

tabMasterRecord.TabVisible := False;
```

2 Add the following public method to the form:

Code	InitDialog Method

```
procedure TfrmMyUserInput.InitDialog(aStartupPage: Boolean; const aDefaultHeading,
          aCompanyName: String);
begin

  if aStartupPage then
    pgCtlMain.ActivePage := tabStartup

  else
    pgCtlMain.ActivePage := tabMasterRecord;
    ebHeading.Text := aDefaultHeading;
    lblCompanyName.Caption := aCompanyName;

end;
```

Create a New Unit

1 Create a new unit and save it in the RBuilder | Demos | 3. EndUser | 1. Report Explorer directory as myRapFuncsForm.pas.

2 Add the following uses clause:

Code	Uses Clause

```
uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs, raFunc,
ppRTTI, myUserInputForm;
```

3 Add the following class declaration:

Code	TmyUserInputFormFunction Class Declaration

```
TmyUserInputFormFunction = class (TraSystemFunction)
public
  class function Category: String; override;

end;
```

4 Implement the Category function with this code.

Code	Category Method

```
class function TmyUserInputFormFunction.Category: String;
begin
  result := 'User Input Forms';

end;
```

5 Add the following TmyUserInputFormFunction descendant:

Code	TmyShowStartupFormFunction Class Declaration

```
TmyShowStartupFormFunction = class(TmyUserInputFormFunction)
  procedure ExecuteFunction(aParams: TraParamList); override;
  class function GetSignature: String; override;

end;
```

6 Implement TmyShowStartupFormFunction with this code.

Code | ExecuteFunction Method

```
procedure TmyShowStartupFormFunction.ExecuteFunction(aParams: TraParamList);
var
  lString: String;
  lInteger: Integer;
  lbResult: Boolean;
  lInputForm: TfrmMyUserInput;
begin

  GetParamValue(0, lString);
  GetParamValue(1, lInteger);

  lInputForm := TfrmMyUserInput.Create(Application);
  try
    lInputForm.InitDialog(True, lString, '');

    if lInputForm.ShowModal = mrOK then
      begin
        lString := lInputForm.ebHeading.Text;

        lInteger := lInputForm.rgDestination.ItemIndex;
        lbResult := True;
      end
    else
      lbResult := False;
  finally
    lInputForm.Free;
  end;

  SetParamValue(0, lString);
  SetParamValue(1, lInteger);
  SetParamValue(2, lbResult);
end;

class function TmyShowStartupFormFunction.GetSignature: String;
begin

  Result := 'function ExecuteStartupForm(var aReportHeading: String; var aDestination-
Index: Integer): Boolean;';

end;
```

7 Add the following TmyUserInputFormFunction descendant:

Code | TmyShowMasterRecordFormFunction Class Declaration

```
TmyShowMasterRecordFormFunction = class(TmyUserInputFormFunction)
  procedure ExecuteFunction(aParams: TraParamList); override;
  class function GetSignature: String; override;
end;
```

8 Implement TmyShowMasterRecordFormFunction with this code.

Code | ExecuteFunction Method

```
procedure TmyShowMasterRecordFormFunction.ExecuteFunction(aParams: TraParamList);
var
  lString: String;
  lBoolean: Boolean;
  lbResult: Boolean;
  lInputForm: TfrmMyUserInput;
begin
  GetParamValue(0, lString);
  GetParamValue(1, lBoolean);

  lInputForm := TfrmMyUserInput.Create(Application);
  try
    lInputForm.InitDialog(False, '', lString);
    lInputForm.cbIncludeAddress.Checked := lBoolean;

    if lInputForm.ShowModal = mrOK then

      begin
        lBoolean := lInputForm.cbIncludeAddress.Checked;
        lbResult := True;
      end
    else
      lbResult := False;
  finally
    lInputForm.Free;
  end;

  SetParamValue(1, lBoolean);
  SetParamValue(2, lbResult);
end;

class function TmyShowMasterRecordFormFunction.GetSignature: String;
begin
  result := 'function ExecuteMasterRecordForm(const aCompanyName: String; var aDisplay-
Address: Boolean): Boolean;';

end;
```

9 Finally, add an initialization section to the unit and add the following two lines:

Code | Function Registration

```
initialization

raRegisterFunction('ExecuteStartupForm', TmyShowStartupFormFunction);
raRegisterFunction('ExecuteMasterRecordForm', TmyShowMasterRecordFormFunction);
```

Add Units to Project

1 Open the RBuilder | Demos | 3. EndUser | 1. Report Explorer | EndUser.dpr project.

2 Select Project | Add to project, and add the two new units, myUserInputForm.pas and myRapFuncsForm.pas.

3 Compile and run the project.

4 Create a new report.

Create a DataView

1 Once the Report Designer is open, click the Data tab.

2 Select File | New | Query Designer and click OK.

3 In the Query Designer Tables tab double-click the Customer table.

4 Double-click the Orders table to open the Join Table dialog and remove the TaxRate join from the Joined Fields list. Then click OK.

5 In the Fields tab check the All Fields check box.

6 Click OK to create the DataView.

Layout the Report

1 Click the Design tab

2 From the Report menu, select Groups... and create a group based on plCustomer.CustNo

3 From the Report menu, select the Title, Header and Footer items to turn on the Title band and turn off the Header and Footer bands.

4 In the Group Header band, add a DBText  , a Region  and two Variables  inside the region.

5 Assign the DBText to the Company field and set it to AutoSize.

6 The two variables inside the region will hold address information so place the region and the variables beneath the DBText component.

7 Turn off the line around the region.

8 In the Title of the report, add 3 Variables  .

9 Variable3 will be the Report title. Center it at the top of the band and set it to 16pt Arial.

10 Variable4 will be our salutation. Place it half an inch below Variable3 against the left margin. Set it to 10pt Times New Roman.

11 Variable5 is the message body. Place it just below Variable 4 against the left margin. Set its width to 7.5" and its height to 0.5". Set WordWrap to True.

12 In the Detail band, add 3 DBText  controls.

13 Place the DBText controls left to right, assigned to OrderNo, SaleDate and AmountPaid.

14 Set the Detail band to Static height and set its height to 0.25".

Declare Global Variables

1 Click the Calc tab.

2 Right-click the Code Explorer's treeview and select Module.

3 Select the Declarations node.

4 Right-click the Variables item in the listview and select New.

5 In the Code Editor, add the following declarations:

```
myHeaderString: String;
myRecipientString: String;
myDescriptionString: String;
```

These variables correspond to the Variables you created in the Title band.

Implement Global OnCreate

1 Click the Events node.

2 Right click the OnCreate item in the listview and select New.

3 In the Code Editor, add the following two local variables to the event handler:

```
var
  liIndex: Integer;
  lbExecuted: Boolean;
```

4 Enter the code on the following page for the event handler.

Basically, in this code we are executing the startup version of the form where we ask for a report heading and for a recipient. liIndex tells us which radio button was selected on the form, so we carefully phrase our salutation accordingly.

Code	Global OnCreate Event

```
begin

  liIndex := 0;
  myHeaderString := 'Monthly Sales Report';

  if ExecuteStartupForm(myHeaderString, liIndex) then
    begin
      case liIndex of
        0:
         begin
     myRecipientString := 'Attn: Marketing Team, ';
          myDescriptionString := '       Here are the monthly figures someone down there
          asked for. Pass this report from desk to desk until someone remembers they
          requested it.';
        end;
        1:
         begin
           myRecipientString := 'My humble apologies, Your Highness,';
           myDescriptionString := '       I regret to interrupt you, but this lowly pro-
          grammer brings you the monthly sales report that you will not like.';
         end;
        2:
         begin
           myRecipientString := 'Dear Brogan, Ethan, Jake, Rebekah and Kathleen,';
          myDescriptionString := '       Since I will soon be joining you as a VP, this
          will be my last monthly report.';
         end;
        3:
         begin
           myRecipientString := 'FILE COPY';
           myDescriptionString := 'Monthly Sales Report';
         end;
      end;
    end
  else
    begin
      myRecipientString := 'Attn: Marketing Team, ';
       myDescriptionString := '       Here are the monthly figures someone down there
      asked for. Pass this report from desk to desk until someone remembers they
      requested it.';
    end;

end;
```

Group Header BeforePrint Event

1 Right click the Code Explorer's treeview and select Events.

2 Scroll down the tree view and find the Group Header node.

3 Click that node, then right-click the BeforePrint event in the listview.

4 Implement this event handler with the code as shown below.

5 Right click the Code Explorer's treeview and select Variables.

6 Click the Title band node.

In the listview you will see the three Variables that reside in the Title band. In the next step, we will assign values for these variables

Code	Group Header BeforePrint Event

```
var
  lDisplayAddress: Boolean;
begin

  lDisplayAddress := True;
  ExecuteMasterRecordForm(plCustomer['Company'], lDisplayAddress);
  Region1.Visible := lDisplayAddress;

end;
```

7 For the Variables, make the following assignments:

Code	Title Band Variable OnCalc Events

Variable3:

```
Value := MyHeaderString;
```

Variable4:

```
Value := MyRecipientString;
```

Variable5:

```
Value := MyDescriptionString;
```

8 Click the Group Header node in the Code Explorer's treeview.

Make the following assignments by enterning the code as shown below.

Code	Title Band Variable OnCalc Events

```
Variable1:

Value := plCustomer['Addr1'] + '  ' +  plCustomer['Addr2'];

Variable2:

Value := plCustomer['City'] + ', ' + plCustomer['Country'] + ' ' + plCustomer['Zip'];
```

Compile and Preview

1 Compile the code by right clicking in the Code Editor and selecting Compile.

2 Now click the Preview tab.

Before anything appears you should see the Startup Form asking you for a heading and a destination (recipient).

3 Enter a value for a title and select a recipient and click ok.

Now, this next dialog will get a little annoying, but this is a demo to make a point, so bear with it.

4 The Master Record dialog will display for each master record group. Click the check box to indicate whether to display the company's address.

That's all. Don't forget that a finished version of this tutorial exists in the RAP demo project as report #35.

APPLICATION TUTORIALS

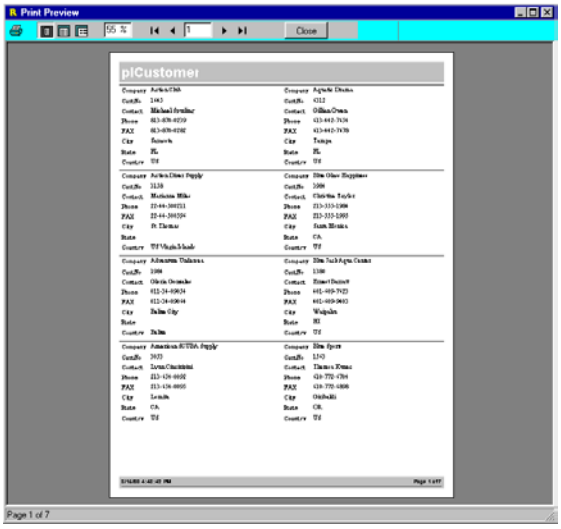
APPLICATION TUTORIALS

Building a Reporting Application

Overview

This tutorial will show you how to do the following:

- Build an architecture for dynamically creating report forms
- Use your own customized print preview form



Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'frmMain'.

3 Select File | Save As from the Delphi menu and save the form under the name rbMain in the My RB Tutorials directory (established on page 181).

4 Select View | Project Manager from the Delphi main menu.

5 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

6 Save the project under the name rbMainProj in the My RB Tutorials directory.

Create a ListBox and Preview Button on the Main Form

1 Select the Standard tab on the Delphi component palette.

2 Add a ListBox to the upper left corner of the Delphi form.

3 Configure the ListBox:

Name	lbxReports
Left	8
Top	8
Width	280
Height	300

4 Create a button below the ListBox.

5 Configure the Button:

Name	btnPreview
Caption	Preview
Left	208
Top	312

6 Set the form size:

Height	368
Width	303

7 Select File | Save from the Delphi main menu.

8 Close the rbMain unit.

Add a Form to the Project

1 Select View | Project Manager from the Delphi main menu.

2 Right-click over rbMainProj in the Project Manager and select the Add... menu option.

3 Locate the rbViaWiz unit in the Tutorials directory. If you have not completed this tutorial, you can find this unit in RBuilder\Tutorials. Add this form to your project.

4 Right-click over the rbMainProj project and access the Options... menu item. The Project Options dialog will be displayed.

5 Select the Forms tab. 'frmViaReportWizard' should be listed in the Auto-create forms list. Select this form and click '>' button to move it to the Available forms list. Close the dialog.

Note: When a form is added to a Delphi project, it is automatically added to the Auto-create list. Forms that appear in this list are created when the application is executed. We will be instantiating this form manually using Object Pascal and do not want the application to create it for us.

Create an Ancestor Form Class

1 Select File | New... from the Delphi main menu.

2 Select the Unit icon and click the OK button. A new unit entitled 'Unit1' will be created.

3 Select File | Save As from the Delphi main menu and save the unit under the name 'rbClass' in the My RB Tutorials directory.

4 Add the following code between the interface and implementation section of the unit:

```
uses
Forms, ppReport;

type

TrbReportForm = class(TForm)
protected
   function GetReport: TppReport; virtual;
   ↳ abstract;

public
   property Report: TppReport read
   ↳ GetReport;

end;
```

Note: A Delphi form is just another class within the VCL. This declaration creates a non-visual descendant of a TForm. It then declares a report property that can be used by any routine referencing the class. Next we will modify a visual form to become a descendant of TrbReportForm. We will then implement the GetReport procedure by returning the report object contained in the form. This technique allows us to easily work with the report objects in TrbReportForm descendants.

5 Select File | Save from the Delphi main menu.

Make the Tutorial Report Form a Descendant of TrbReportForm

1 Select View | Project Manager from the Delphi main menu.

2 Double-click on the rbViaWiz unit to open the form.

3 Access the rbViaWiz unit and maximize the Code Editor.

4 Add rbClass to the uses clause at the top of the unit. The code should look like this:

```
unit rbViaWiz;

interface

uses
   Windows, Messages, SysUtils, Classes,
   Graphics, Controls, Forms, Dialogs,
   StdCtrls, ppBands, ppCtrls, ppPrnabl,
   ppClass, ppDB, ppProd, ppReport,
   ppComm, ppCache, ppDBPipe, ppDBBDE, Db,
   DBTables, ppVar, rbClass;
```

Note: We must add the rbClass unit to the uses clause so that we can descend from the TrbReportForm.

5 Change the inherited form name from TForm to TrbReportForm. The first line of the form class declaration should look like this:

```
type
   TfrmViaReportWizard =
   ↳ class(TrbReportForm)
```

6 Click on the rbClass tab in the Code Editor.

7 Copy the protected section of the class (including the protected keyword) into your clipboard. Here is the code you need to copy:

```
protected
   function GetReport: TppReport; virtual;
   ↳ abstract;
```

8 Click on the rbViaWiz tab in the Code Editor.

9 Scroll down until you locate the public keyword at the bottom of class declaration for the form.

10 Insert the contents of your clipboard above the public keyword. The declaration should now look like this:

```
protected
   function  GetReport: TppReport; vir-
   tual;
   ↳ abstract;
public
{ Public declarations }
end;
```

11 Replace the virtual and abstract keywords with override so the declaration looks like this:

```
protected
   function GetReport: TppReport;
   override;
public
{ Public declarations }
end;
```

12 Copy the GetReport function declaration into the clipboard and scroll down to the implementation section of the unit.

13 Paste the declaration into the implementation section, remove the override keyword, and add the following code so that the function looks like this:

```
implementation
{$R *.DFM}

function TfrmViaReportWizard.GetReport:
TppReport;
begin
   Result := rbCustomerList;
end;
```

Note: This routine will be called whenever the rbReportForm.Report property is referenced.

14 Select File | Save from the Delphi main menu.

15 Close the Code Editor.

Populate the List Box in the OnCreate Event of the Main Form

1 Select View | Project Manager from the Delphi main menu.

2 Double-click on rbMain to open this form.

3 Select the Events tab of the Object Inspector.

4 Double-click the OnCreate event. The event handler shell will be generated and the Code Editor will be displayed.

5 Add the following line of code to the event handler:

```
lbxReports.Items.AddObject('Creating a
Report via the Report Wizard',
TObject(TfrmViaReportWizard));
```

Note: This code stores a description of the report and the class type for the form containing the report in the list. The class type will be used to instantiate an instance of the form.

Code the LaunchReport Procedure in the Main Form

1 Add the LaunchReport procedure declaration to the private section of the main form. The code should look like this:

```
private
   procedure LaunchReport;
public
{ Public declarations }
end;
```

2 Add the following units to the uses clause at the top of the unit:

```
ppReport, rbViaWiz, rbClass
```

3 Add the LaunchReport procedure to the implementation section. The code should look like the code below.

4 Select File | Save from the Delphi main menu.

Note: This routine retrieves the class type from the list box, instantiates a form of that type, retrieves the report from the form, and then prints the report. This routine assumes two things: that the Report.ModalPreview property is True and that the class type in the ListBox is a TrbReportForm descendant. As you can see, the abstract TrbReportForm class comes in quite handy. As we go forward adding more reports to this project, we can make the forms descendants of this class and add the description and class type to the list box, and the reports will work 'automatically'.

Hook the LaunchReport Procedure to the ListBox and Preview Button

1 Click the Toggle Form/Unit icon on the Delphi View toolbar (or hit the F12 key). You should see the main form.

2 Select the ListBox control and then double-click on the OnDblClick event in the Object Inspector.

3 Add a call to LaunchReport to this event handler. The code should look like this:

```
procedure TfrmMain.lbxReportsDlClick(Sender:
↳ TObject);
begin
   LaunchReport;
end;
```

4 Select the Preview button on the form and then double-click on the OnClick event in the Object Inspector.

5 Add a call to LaunchReport to this event handler.

6 Select File | Save from the Delphi main menu.

7 Select Project | Compile rbMain. Fix any compilation problems.

8 Close the Code Editor and the rbMain form.

```
procedure TfrmMain.LaunchReport;
var
   lFormClass: TFormClass;
   liIndex: Integer;
   lForm: TForm;
   lReport: TppReport;
begin

   liIndex := lbxReports.ItemIndex;

   if (liIndex = -1) then Exit;

   lFormClass := TFormClass(lbxReports.Items.Objects[liIndex]);

   lForm := lFormClass.Create(Application);

   lReport := TrbReportForm(lForm).Report;

   lReport.Print;

   lForm.Free;

end;
```

9 Run the application. The list box should contain one item. If you double-click on the item or select the item and click the Preview button, you should see the report in the Print Preview window.

Create a Customized Print Preview Form

1 Close the application and return to the Delphi development environment.

2 Select File | Open from the Delphi main menu. Locate the ppPrvDlg unit in the RBuilder\Source directory. Open this unit.

3 In the Object Inspector, change the form name from ppPrintPreview to rbPrintPreview. (You may have to use the drop-down list box at the top of the Object Inspector to find the ppPrintPreview form).

4 Save the form in the My RB Tutorials directory under the name rbPrvDlg.

5 Select the panel at the top of the form (named pnlPreviewBar).

6 Set the color to clAqua.

7 Bring the Code Editor to the front and scroll to the very bottom of the unit and check the initialization section of the unit. The code should look like this:

```
initialization
  ppRegisterForm(TppCustomPreviewer,
  ↪ TrbPrintPreview);
finalization
  ppUnRegisterForm(TppCustomPreviewer);
end.
```

Note: This register call replaces the built-in ReportBuilder Print Preview form with this custom version. The initialization section fires when the unit is loaded. The unit is loaded only if it appears in the uses clause of another unit or is included in the project. For this reason, you should always add your preview form to your project or add the form's unit name to the uses clause of your application's main unit.

8 Select File | Save from the Delphi main menu.

9 Close the Code Editor.

Add the Customized Print Preview Form to your Project

1 Select the View | Project Manager option from the Delphi main menu.

2 Right-click over rbMainProj and select the Add... menu option.

3 Locate rbPrvDlg.pas in the My RB Tutorials directory and add it to the project.

4 Right-click over rbMainProj and select the Options... menu item.

5 Select the Forms tab and move the rbPrintPreview form from the Auto-create list to the Available forms list.

6 Select Project | Compile rbMainProj. Fix any compilation problems.

7 Select File | Save from the Delphi main menu.

8 Run the application. When you preview the report, you should see the customized form instead of the standard ReportBuilder Print Preview form.

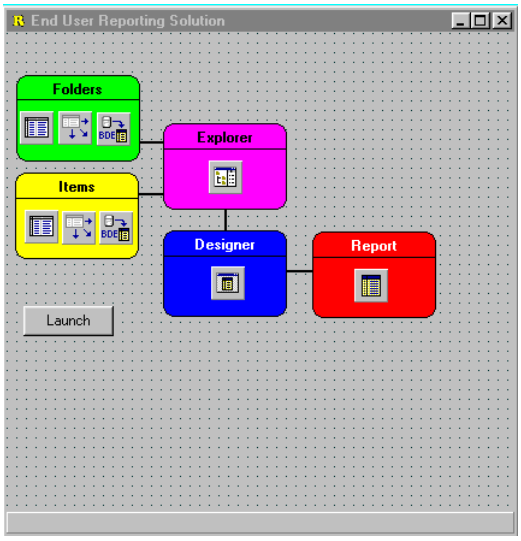
Note: You can add all of the tutorial reports to this project in the same way that we added this one. In order to see a version of this application that includes all of the tutorial reports, open the rbMainProj project in the RBuilder\Tutorials directory.

Building an End-User Reporting Application

Overview

This tutorial will show you how to do the following:

- Combine ReportBuilder components into an end-user reporting solution
- Configure Delphi data access objects for use with the Report Explorer
- Use the Report Designer and Report Explorer components



Create the Folder Table

1 Select Tools | Database Desktop from the Delphi main menu.

2 Select File | Working Directory from the Database Desktop main menu.

3 Click the Browse button and select the following directory:

C:\Program Files\Common Files\Borland Shared\Data

4 Click the OK button to close the dialog.

5 Select File | New | Table.

6 The Create Table dialog will be displayed. It should default to a table type of Paradox 7. Click OK to continue the table creation process.

7 Enter the following fields in the 'Create Paradox Table' dialog:

FieldName	Type	Size	Key
FolderId	+		*
Name	A	60	
ParentId	I		

Note: The + symbol in the Type of the FolderId field indicates that this is an AutoIncrement field. The asterisk indicates that it is also the key field. The Name field is Alpha 60. The ParentId field is an integer.

Note: Paradox tables are used here because they are accessible to all users of Delphi. You can create these tables in any database format you prefer. For an example of a database-specific end-user application for your database, see RBuilder\Demos\EndUser Databases.

8 Locate the Table properties drop-down list on the right corner of the dialog.

9 Expand this list and select the Secondary Indexes option.

10 Click the Define button. Add the ParentId field to the Indexed fields list and click the OK button. Name the index ixParentId. Click OK.

11 Click the Save As button and locate the directory where your DBDEMOS alias is pointed (usually C:\Program Files\Common Files\Borland Shared\Data). Save the table under the name rbFolder.

Create the Item Table

1 Select File | New | Table. Accept the Paradox 7 table type. Enter the following fields:

FieldName	Type	Size	Key
ItemId	+		*
FolderId	I		
Name	A	60	
Size	I		
ItemType	I		
Modified	@		
Deleted	@		
Template	B		

Note: The + symbol in the Type of the ItemId field indicates that this an AutoIncrement field. The asterisk indicates that it is also a key field. The Name field is Alpha 60. The FolderId field is an integer. The @ symbol used for the Modified and Deleted fields indicates a datatime type. The B on the template field indicates binary. If you set the Report.Template.Format to ftASCII, you can use a memo field instead. The ItemType field contains an integer indicating whether the item is a report, data module, or code module.

2 Locate the Table properties drop-down list on the right corner of the dialog.

3 Expand this list and select the Secondary Indexes option.

4 Click on the Define button. Add the FolderId, ItemType, and Name fields to the Indexed fields list and click the OK button. Name the index ixFolderIdItemTypeName.

5 Click the Save As... button and locate the directory where your DBDEMOS alias is pointed (usually C:\Program Files\Common Files\Borland Shared\Data) and save the table under the name rbItem.

6 Close Database Desktop.

Create a New Application

1 Select File | New Application from the Delphi menu. This will create a new project and a blank form.

2 Set the Form name to 'myEndUserSolution'.

3 Set the form's Caption to End-User Reporting Solution.

4 Select File | Save As from the Delphi menu and save the form under the name rbEndUsr in the My RB Tutorials directory (established on page 181).

5 Select File | Save Project As.

6 Save the project under the name rbEUProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create Data Access Components for the Folder Table

1 Select the Data Access tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblFolder
TableName	rbFolder.db

4 Add a DataSource component to the form.

5 Configure the DataSource component:

DataSet	tblFolder
Name	dsFolder

6 Select the RBuilder tab of the Delphi component palette.

7 Add a DBPipeline component  to the form.

8 Configure the DBPipeline component:

DataSource	dsFolder
Name	plFolder
Visible	False

Note: The Visible property of a DataPipeline controls whether the data pipeline can be selected within the Report Designer and assigned to a report or detail band. Since all pipelines created in this tutorial will be used to store supporting information, none should be visible within the Report Designer.

Create Data Access Components for the Item Table

1 Add a Table component to the form.

2 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblItem
TableName	rbItem.db

3 Add a DataSource component to the form.

4 Configure the DataSource component:

DataSet	tblItem
Name	dsItem

5 Select the RBuilder tab of the Delphi component palette.

6 Add a DBPipeline component  to the form.

7 Configure the DBPipeline component:

DataSource	dsItem
Name	plItem
Visible	False

Create the ReportBuilder Components

1 Add a Report component  to the form.

2 Add a Designer component  to the form.

3 Set the Report property of the Designer to ppReport1.

4 Add a ReportExplorer component  to the form.

5 Set the Designer property of the ReportExplorer to ppDesigner1.

6 Configure the ReportExplorer component:

FolderPipeline	plFolder
ItemPipeline	plItem

Configure the Designer Component

1 Select the Designer component.

2 Expand the DataSettings property in the Object Inspector.

3 Set the DatabaseName property to DBDEMOS.

Note: This property specifies the database that will be queried.

4 Set the SQLType to sqBDELocal.

Note: Since we're using Local SQL via the BDE, we need to set the SQL type. If you are not using Local SQL to access Paradox and dBase tables, you should use an SQLType of sqSQL1, since the Local SQL format is not compatible with most other databases.

5 Add daDatMan, daDBBDE to the uses clause of the form's unit.

6 Select File | Save from the Delphi main menu.

Note: The daDatMan unit contains a data manager class that makes the data workspace available from within the Report Designer. The daDBBDE unit associates a BDE-based dataview class with the Query Wizard and Query Designer. Therefore, these tools will create instances of this dataview when invoked. This design allows other query dataview classes to be created and associated with these tools.

Compile and Run the Application

1 Select the Win32 tab of the Delphi component palette.

2 Add a StatusBar to the form.

3 Configure the StatusBar:

Name	stbStatusBar
SimplePanel	True

4 Add a Button component to the form.

5 Configure the Button:

Name	btnLaunch
Caption	Launch

6 Add the following code to the OnClick event of the button:

```
if not(ppReportExplorer1.Execute) then
   stbStatusBar.SimpleText :=
   ↳ ppReportExplorer1.ErrorMessage;
```

Note: This code calls the Execute method to invoke the Report Explorer. If we have not configured one of the Report Explorer properties correctly, the form will not be displayed, False will be returned, and the reason for the failure will be displayed in our status bar.

7 Select Project | Compile rbEUProj. Fix any compilation problems.

8 Select File | Save from the Delphi main menu.

9 Run the Project.

10 Click the Launch button. The Report Explorer should be displayed. If the Report Explorer is not displayed, an explanation will be displayed in the panel at the bottom of the window.

Create New Folders

1 Select File | New | Folder from the Report Explorer main menu.

2 Type Marketing in the folder name and press the Enter key.

3 Select the main folder (entitled All Folders).

4 Right-click over the white space of the Folder Tree and select the New Folder option from the popup menu.

5 Type Accounting in the folder name and press the Enter key.

6 Right-click over the white space of the Folder Tree and select the New Folder option. A folder should be created under the Accounting folder.

7 Type 1998 in this folder and press the Enter key.

8 Right-click over the 1998 folder and select the New Report menu option.

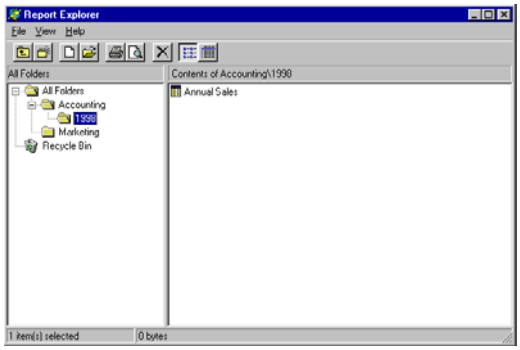
9 Select the File | Save As from the Report Designer main menu and save the report under the name Annual Sales.

10 Close the Report Designer.

11 The 'Annual Sales' report should be displayed in the 1998 folder.

Note: You can create and configure reports from this point, just as you would from within Delphi. And as you can see, the Report Explorer is quite easy to use due to its similarity to the Windows Explorer.

The running application should look like this:

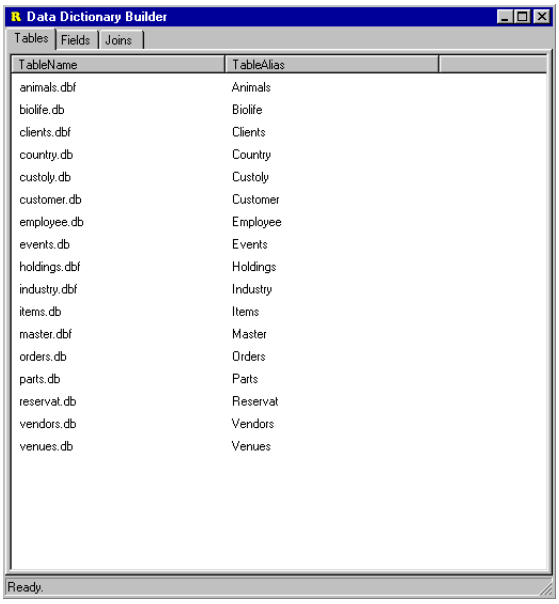


Adding Data Dictionary Support to the End-User Application

Overview

This tutorial will show you how to do the following:

- Configure Delphi data access objects for use with the Data Dictionary
- Configure the Data Dictionary component
- Use the Query Wizard within the Report Designer



Create the Field Data Dictionary Table

1 Select Tools | Database Desktop from the Delphi main menu.

2 Select File | Working Directory from the Database Desktop main menu.

3 Click the Browse button and select the following directory:

C:\Program Files\Common Files\Borland Shared\Data

4 Click the OK button to close the dialog.

5 Select File | New | Table.

6 The Create Table dialog will be displayed. It should default to a table type of 'Paradox 7'. Click OK to continue the table creation process.

7 Enter the following fields in the dialog:

FieldName	Type	Size	Key
TableName	A	60	*
FieldName	A	60	*
FieldAlias	A	60	
Selectable	A	1	
Searchable	A	1	
Sortable	A	1	
DataType	A	60	
AutoSearch	A	1	
Mandatory	A	1	

Note: The TableName and FieldName fields make up the primary key. The FieldAlias is the value that will be displayed to the end user. The FieldName is the raw field name from the database table. The TableName is the raw table name of the database table.

8 Click the Save As button and locate the directory where your DBDEMOS alias is pointed (usually C:\Program Files\Common Files\Borland Shared\Data) and save the table under the name rbField.

Create the Table Data Dictionary Table

1 Select File | New | Table. Accept the Paradox 7 table setting. Enter the following fields in the Create Paradox 7 Table dialog:

FieldName	Type	Size	Key
TableName	A	60	*
TableAlias	A	60	

Note: The TableName represents the raw table name of the database table. The TableAlias will be displayed to the end user using dot notation (i.e. TableAlias.FieldAlias).

Note: This tutorial uses Paradox tables simply because most Delphi developers have access to this type of table. You can use whatever database format you desire when creating these tables.

2 Click the Save As button and locate the directory where your DBDEMOS alias is pointed (usually C:\Program Files\Common Files\Borland Shared\Data) and save the table under the name rbTable.

Create the Join Data Dictionary Tables

1 Select File | New | Table. Accept the Paradox 7 table setting. Enter the following fields in the dialog:

FieldName	Type	Size
TableName1	A	60
TableName2	A	60
JoinType	A	60
FieldNames1	A	255
FieldNames2	A	255
Operators	A	60

2 Click the Save As button and locate the directory where your DBDEMOS alias is pointed (usually C:\Program Files\Common Files\Borland Shared\Data) and save the table under the name rbJoin.

3 Close Database Desktop.

Open the End-User Application

1 Select File | Open Project from the Delphi menu.

2 Open the rbEUProj project from the previous tutorial. If you have not completed this tutorial, you can create a new project and add the rbEndUsr form from the RBuilder\Tutorials directory.

3 Open the rbEndUsr form and save it under the name 'rbEUDD' in the My RB Tutorials directory (established on page 181).

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

4 Select File | Save Project As from the Delphi menu and save the project under the name rbEUDDProj in the My RB Tutorials directory.

Create Data Access Components for the Tables

1 Select the Data Access tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblTable
TableName	rbTable.db

4 Add a DataSource component to the form.

5 Configure the DataSource component:

DataSet	tblTable
Name	dsTable

6 Select the RBuilder tab of the Delphi component palette.

7 Add a DBPipeline component  to the form.

8 Configure the DBPipeline component:

DataSource	dsTable
Name	plTable
Visible	False

Create Data Access Components for the Field Table

1 Add a Table component to the form.

2 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblField
TableName	rbField.db

3 Add a DataSource component to the form.

4 Configure the DataSource component:

DataSet	tblField
Name	dsField

5 Select the RBuilder tab of the Delphi component palette.

6 Add a DBPipeline component to the form.

7 Configure the DBPipeline component:

DataSource	dsField
Name	plField
Visible	False

Create Data Access Components for the Join Table

1 Add a Table component to the form.

2 Configure the Table component:

DatabaseName	DBDEMOS
Name	tblJoin
TableName	rbJoin.db

3 Add a DataSource component to the form.

4 Configure the DataSource component:

DataSet	tblJoin
Name	dsJoin

5 Select the RBuilder tab of the Delphi component palette.

6 Add a DBPipeline component to the form.

7 Configure the DBPipeline component:

DataSource	dsJoin
Name	plJoin
Visible	False

Create the Data Dictionary Component

1 Add a DataDictionary component to the form.

2 Configure the DataDictionary component:

AutoJoin	True
BuilderSettings.DatabaseName	DBDEMOS
FieldPipeline	plField
JoinPipeline	plJoin
TablePipeline	plTable

3 Select File | Save from the Delphi main menu.

Populate the Tables

1 Double-click on the DataDictionary component to launch the Data Dictionary Builder.

2 Right-click over the white space of the Tables tab and select Generate. The tables will generate.

3 Click the Fields tab and then generate the entries.

4 Click the Joins tab. Right-click and select the Add menu option. The join panel will appear.

5 Select Customer.db from the table list on the left.

6 Select Left Outer from the Join Type drop-down list.

7 Select Orders from the table list on the right. The suggested join fields will appear in the Joined Fields section of the panel.

8 Double-click on TaxRate to remove it from the join selection.

9 Click Save.

10 Right-click and select Add. The join panel will appear.

11 Select Orders.db from the table list on the left.

12 Select Left Outer from the Join Type drop-down list.

13 Select Employee from the with table list on the right. The suggested join fields will appear in the Joined Fields section of the panel.

14 Click Save.

15 Close the Data Dictionary Builder.

Configure the Designer Component

1 Select the Designer component  on the form.

2 Expand the DataSettings property in the Object Inspector.

3 Set the DatabaseName property to DBDEMOS.

Note: The DatabaseName indicates the database that contains the data to be retrieved by the reports. In order to keep this tutorial simple, we have used the same database alias for the data dictionary and report data tables. This is not a requirement. You could store the data dictionary in local tables and build queries over a remote database.

4 Set the DataDictionary property to ppDataDictionary1.

5 Set the UseDataDictionary property to True.

Note: Setting both of these properties allows the Designer to display the TableAlias and FieldAlias values from the data dictionary tables instead of the raw field names. Also the Selectable, Searchable and Sortable field values will be used to determine if a particular field can appear on the Select Field, Search Criteria and Set Order pages of the Query Wizard and Query Designer.

6 Select File | Save from the Delphi main menu.

Compile and Run the Application

1 Select Project | Compile rbEUDDProj. Fix any compilation problems.

2 Select File | Save from the Delphi main menu.

3 Run the Project.

Create a Simple Query

1 Create a new report.

2 Select the Data tab of the Report Designer.

3 Select File | New.

4 Select the Query Wizard.

5 Select the Customer table from the list of available tables. Scroll down and select the Orders table.

6 Click the Next button.

7 Click the Choose Fields option.

8 Select Company, Contact, State, Order No., and AmountPaid and move them to the Selected Fields list.

9 Skip the next four pages by clicking Next until you reach the last page.

10 Click the Finish button. The data will be displayed.

11 Close the Data Preview dialog.

12 The Dataview will appear in the workspace of the Data tab.

13 Click on the Design tab.

Create a Simple Report

1 Select File | New from the Report Designer main menu.

2 Select the Report Wizard.

3 Select all of the available fields.

4 Select the remaining options based on your preferences.

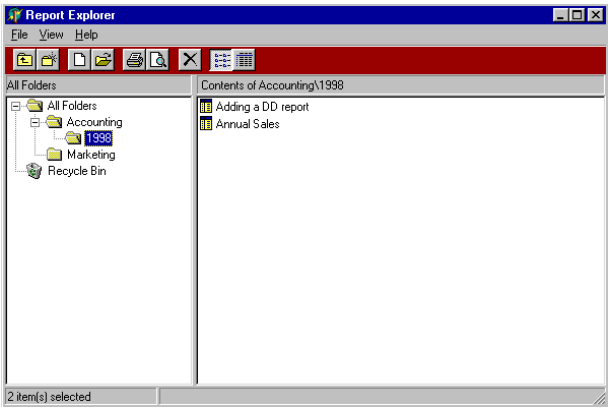
5 After you've clicked the Finish button and returned to the Design tab, save your report under the name 'Orders.'

Customizing the Report Explorer Form

Overview

This tutorial will show you how to do the following:

- Replace the Report Explorer form with a form of your own
- Use the ppRegisterForm procedure



Open the End-User Application

1 Select File | Open Project from the Delphi menu.

2 Open the rbEUProj project from the "Building an End-User Application" tutorial. If you have not completed this tutorial, you can create a new project and open the rbEndUsr form from the RBuilder\Tutorials directory.

3 Close the rbEndUsr form.

4 Select File | Save Project As from the Delphi menu and save the project in the My RB Tutorials directory (established on page 181) under the name rbREProj.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create a New Report Explorer Form

1 Select File | Open... menu from the Delphi main menu.

2 Locate the ppExpFrm.pas unit in the RBuilder\Source directory.

3 Open this form.

4 Change the form name from ppReportExplorer-Form to rbReportExplorerForm.

5 Select the panel at the top of the form (named tbrExplorer) and use the Object Inspector to set the color to clMaroon.

6 Scroll to the bottom of the form unit and inspect the initialization section. It should contain the following:

```
initialization
  ppRegisterForm(TppCustomReportExplorer,
  ↳ TrbReportExplorerForm);
finalization
ppUnRegisterForm(TppCustomReportExplorer);
end.
```

Note: The Delphi Code Editor automatically changed the class name of the form from ppReportExplorerForm to rbReportExplorerForm when you changed the form name in the Object Inspector. This call will replace the 'official' explorer form with the new form we've created.

Note: In order for the initialization section of this new form unit to fire (thus calling the ppRegisterForm routine), we must include this unit in the project. If we do not include this unit in the project, then the standard report explorer will be used.

7 Save the form under the name rbExpFrm in the My RB Tutorials directory.

8 Close the form.

9 Select View | Project Manager from the Delphi main menu.

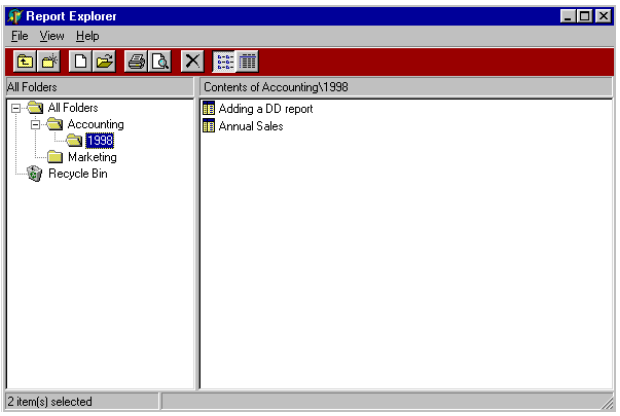
10 Add rbExpFrm to the project and remove the form from the project's auto-create list (Options | Forms tab).

11 Select File | Save from the Delphi main menu.

12 Run the application. When you click the Launch button you should see a maroon toolbar instead of a silver one (indicating that the custom explorer form is being used).

Note: As you can see, you now have total control over the look, feel, and behavior of the Report Explorer form. If you review the source of this form, you'll see that it is calling various functions in the Report Explorer component. These functions are documented in the help.

The customized run-time application should look like this:

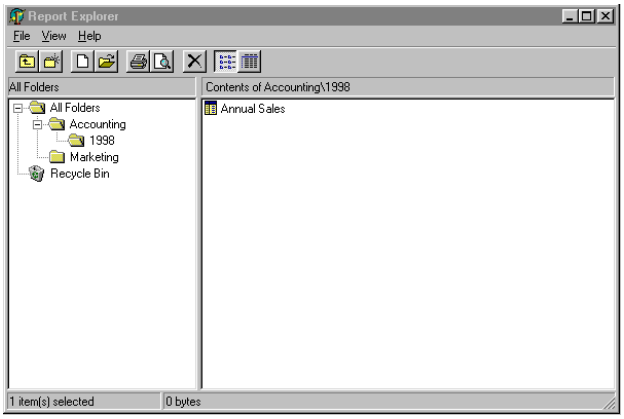


Building a Report Application using InterBase

Overview

This tutorial will show you how to do the following:

- Configure Delphi data access objects for use by the Report Explorer
- Create the InterBase tables required by the Report Explorer
- Use the Report, Report Designer, and Report Explorer components
- Create the InterBase tables required by the Data Dictionary (optional)



Review the Create Tables SQL script for the Report Explorer

Note: This tutorial requires that you have InterBase installed and know how to use it. The concepts covered in this tutorial can be adapted to other SQL based databases. The SQL scripts used to create the InterBase tables can likewise be used as a starting point for creating scripts for other types of SQL databases. However, examples for a number of popular database products are included with ReportBuilder. These examples can be found in RBuilder\Demos\EndUser Databases.

Note: The Report Explorer is connected to InterBase tables in this tutorial. Thus, the folders and reports created by the end user are stored in InterBase tables. The Report Designer is also connected to InterBase to enable the end user to use the Query Wizard to build data views. It is also possible to connect the Report Explorer and Report Designer to separate databases. For some applications it may make sense to connect the Report Explorer to local database tables and connect the Report Designer to a database server such as InterBase. Under this scenario each end user would have his own set of folders and reports.

1 Select File | Open from the Delphi main menu to access the Open dialog.

2 Select the Any File (*.*) option from the Files of Type list at the bottom of the dialog.

3 Locate the RBuilder\Tutorials directory and open the ExploreTables.sql file.

4 Find the Connect statement at the top of the file and review the database, user name, and password parameters. The database name is configured to access the employee demo database that is installed with InterBase. Modify the parameters as needed to reflect your InterBase installation.

```
CONNECT "C:\Program Files\Common
↳ Files\Borland Shared\Data\employee.gdb"
↳ USER "SYSDBA" PASSWORD "masterkey";
```

5 Review the create statements for the folder table. The table is created first, followed by two indexes. Next a generator and a trigger are created for the folder_id value.

Note: Do not modify the table or index definitions. Do not add field constraints such as NOT NULL or UNIQUE. Do not define a primary key. The Report Explorer is designed to support desktop and client/server databases. This requirement limits the flexibility of the table definitions. Adding field constraints such as those described above will result in Delphi run-time errors in your end-user reporting application.

Code *The SQL for creating the Folder table:*

```
/* create Folder table */

CREATE TABLE rb_folder
(folder_id INTEGER,
name VARCHAR(60),
parent_id INTEGER);

CREATE INDEX folder_idx ON rb_folder (folder_id);

CREATE INDEX parent_idx ON rb_folder (parent_id);

COMMIT;

/* create generator for folder id */

CREATE GENERATOR folder_id_gen;
SET GENERATOR folder_id_gen to 1;

/* create trigger to add unique folder id */

SET TERM !! ;
CREATE TRIGGER set_folder_id FOR rb_folder
BEFORE INSERT
AS
BEGIN
      new.folder_id = gen_id(folder_id_gen, 1);
END !!
SET TERM ; !!
COMMIT;
```

6 Review the create statements for the item table. The table is created first, followed by two indexes. Next, a generator and a trigger are created for the item_id value.

Run the Create Tables SQL Script

1 Select Database | Explore from the Delphi main menu to access the Database Explorer

2 Double-click the database alias corresponding to your Interbase database. The alias for a default installation of the local InterBaser server is IBLOCAL.

3 Enter 'masterkey' as the password and click OK.

4 Right-click to access the popup menu for the database. Select 'ISQL' to access the InterBase ISQL utility.

5 Select 'File | Run an ISQL Script' from the main menu of the InterBase ISQL utility. Open the ExploreTables.sql file. A 'Save output to a file?' dialog will be displayed.

6 .Click the No button.

Code *The SQL for creating the Item table*

```
/* create Item table */
CREATE TABLE rb_item
(item_id INTEGER,
folder_id INTEGER,
name VARCHAR(60),
item_size INTEGER,
item_type INTEGER,
modified FLOAT,
deleted FLOAT,
template BLOB SUB_TYPE 0 SEGMENT SIZE 400);

CREATE INDEX item_idx ON rb_item (item_id);

CREATE INDEX folder_item_name_idx ON rb_item (folder_id, item_type, name);

COMMIT;

/* create generator for item id */

CREATE GENERATOR item_id_gen;
SET GENERATOR item_id_gen to 1;

/* create trigger to add unique item id */

SET TERM !! ;
CREATE TRIGGER set_item_id FOR rb_item
BEFORE INSERT
AS
BEGIN
      new.item_id = gen_id(item_id_gen, 1);
END !!
SET TERM ; !!
COMMIT;
```

Note: The SQL script should connect to the database and create the database objects. If you receive a message indicating the script completed with errors, then review the output information - error messages are displayed following the statement in which the error occurred.

7 Close the InterBase ISQL window.

8 Close the Database Explorer.

Create a New Application

1 Select File | New Application from the Delphi main menu. This will create a new project and blank form.

2 Set the Form name to 'frmEndUserInterBase'.

3 Set the form's Caption to End-User Reporting Solution.

4 Select File | Save As from the Delphi menu and save the form under the name rbEUIB in the My RB Tutorials directory (established on page 181).

5 Select View | Project Manager from the Delphi main menu.

6 Right-click over the project name in the Project Manager (usually Project1.exe) and select the Save menu option.

7 Save the project under the name rbEUIBProj in the My RB Tutorials directory.

Note: Save your work frequently to avoid losing changes should your computer unexpectedly lose power or crash.

Create Data Access Components for the Folder Table

1 Select the Data Access tab of the Delphi component palette.

2 Add a Database component to the form.

3 Configure the Database component:

AliasName	IBLocal
DatabaseName	rbData
Name	rbData
LoginPrompt	False
Params	USER NAME=SYSDBA PASSWORD=masterkey

4 Test the database connection by setting the Connected property of the Database component to True. Resolve any errors and reset the property to False.

5 Add a Table component to the form.

6 Configure the Table component:

DatabaseName	rbData
Name	tblFolder
TableName	RB_FOLDER

7 Add a DataSource component to the form.

8 Configure the DataSource component:

DataSet	tblFolder
Name	dsFolder

9 Select the RBuilder tab of the Delphi component palette.

10 Add a DBPipeline component  to the form.

11 Configure the DBPipeline component:

DataSource	dsFolder
Name	plFolder
RefreshAfterPost	True
Visible	False

Note: We set the DBPipeline.Visible property to False so that the Report Designer will not display this data pipeline.

Create Data Access Components for the Item Table

1 Select the Data Access tab of the Delphi component palette.

2 Add a Table component to the form.

3 Configure the Table component:

DatabaseName	rbData
Name	tblItem
TableName	RB_ITEM

4 Add a DataSource component to the form.

5 Configure the DataSource component:

DataSet	tblItem
Name	dsItem

6 Select the RBuilder tab of the Delphi component palette.

7 Add a DBPipeline component  to the form.

8 Configure the DBPipeline component:

DataSource	dsItem
Name	plItem
RefreshAfterPost	True
Visible	False

Create the ReportBuilder Components

1 Add a Report component  to the form.

2 Add a Designer component  to the form.

3 Set the Report property of the Designer to ppReport1.

4 Configure the Designer component:

DataSettings.DatabaseName	rbData
DataSettings.SQLType	sqSQL2
Database Type	dtInterbase

Note: The DataSettings property of the Designer contains a number of properties used to control the database connection and data access. The SQLType should be set to sqSQL2 for database servers and to sqlBDELocal for desktop databases such as Paradox and dBase.

5 Add a ReportExplorer component to the form.

6 Set the Designer property of the ReportExplorer to ppDesigner1.

7 Configure the ReportExplorer component:

FolderPipeline	plFolder
FolderFieldNames.FolderId	FOLDER_ID
FolderFieldNames.Name	NAME
FolderFieldNames.ParentId	PARENT_ID
ItemPipeline	plITEM
ItemFieldNames.Deleted	DELETED
ItemFieldNames.FolderId	FOLDER_ID
ItemFieldNames.ItemId	ITEM_ID
ItemFieldNames.ItemType	ITEM_TYPE
ItemFieldNames.Modified	MODIFIED
ItemFieldNames.Name	NAME
ItemFieldNames.Size	ITEM_SIZE
ItemFieldNames.Template	TEMPLATE

Note: For simplicity, we've defined our database tables so that the table field names correspond closely with the ItemFieldName and FolderFieldName properties. The exception is the item_size field, which is assigned to the ItemFieldNames.Size property. 'Size' is a reserved word in InterBase and therefore cannot be used as a field name.

Note: While it is acceptable to name the fields differently in your database table, it is not acceptable to change the data types or other field attributes, as the Report Explorer relies on these names.

Run the Application

1 Select the Win32 tab of the Delphi component palette.

2 Add a StatusBar to the form.

3 Configure the StatusBar component:

Name	stbStatusBar
SimplePanel	True

4 Add a Button component to the form.

5 Configure the Button component:

Name	btnLaunch
Caption	Launch

6 Add the following code to the OnClick event handler of the button:

```
Session.SQLHourGlass := False;

if not(ppReportExplorer1.Execute) then
   stbStatusBar.SimpleText:=
   ↳ ppReportExplorer1.ErrorMessage;
```

Note: First we set the SQLHourGlass property to False. We do not want the cursor to be displayed when using the Report Explorer to create folders and reports. Next, we called the Execute method to invoke the Report Explorer. If we have not configured one of the Report Explorer properties correctly, the form will not be displayed and False will be returned. The reason for the failure will be displayed in the 'status bar.'

7 Add the following units to the 'uses' clause at the top of the form unit: daDatMan, daDBBDE.

Note: The daDatMan unit contains a data manager class that makes the data workspace available from within the Report Designer. The daDBBDE unit associates a BDE-based dataview class with the Query Wizard and Query Designer. Therefore, these tools will create instances of this dataview when invoked. This design allows other query dataview classes to be created and associated with these tools.

8 Select Project | Compile rbEUIBProj. Fix any compilation problems.

9 Select File | Save from the Delphi main menu.

10 Run the Project.

11 Click the Launch button. The Report Explorer should be displayed.

Note: If the Report Explorer is not displayed, an explanation will be given in the panel at the bottom of the window.

Create New Folders

1 Select File | New | Folder from the Report Explorer main menu.

2 Type Marketing in the folder name and press the Enter key.

3 Select the main folder (entitled All Folders).

4 Right-click over the white space of the Folder Tree and select the New Folder option from the popup menu.

5 Type Accounting in the folder name and press the Enter key.

6 Right-click over the white space of the Folder Tree and select the New Folder option. A folder should be created under the Accounting folder.

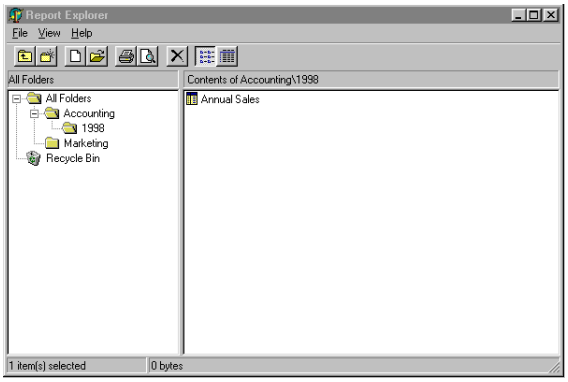
7 Type 1998 in this folder and press the Enter key.

8 Right-click over the 1998 folder and select the New Report menu option.

9 Select File | Save As from the Report Designer main menu and save the report under the name Annual Sales.

10 Close the Report Designer. The Annual Sales report should be displayed in the 1998 folder.

The application should look like this after saving the report:



Note: You can create and configure reports from this point, just as you would from within Delphi. And as you can see, the Report Explorer is quite easy to use due to its similarity to the Windows Explorer.

Note: Using the Data Dictionary is optional. The SQL script for adding the Data Dictionary tables is called DataDictTables.sql and is located in the RBuilder\Tutorials directory. You can refer to the tutorial called “Adding Data Dictionary Support to the End-User Application” for additional information on configuring the relevant components and properties.

APPENDIX

Where Do I Go From Here?

ReportBuilder has many other resources that can aid beginners and proficient users alike.

The Digital Metaphors Website

The material below, with the exception of the help files, can be found at:

www.digital-metaphors.com

Learning ReportBuilder

After you build your end-user application and are ready for distribution, you can send your end users to our website to download Learning ReportBuilder, which is a complete learning system devised for end users. It comes with a stand-alone application, a database, a 125-page series of tutorials in PDF format, and a help file complete with a glossary.

ReportBuilder Help

RB Help is accessible from the Delphi main menu or the Report Designer main menu. You can also select an object and press F1 for help on that object. The RB Help file includes installation information and documents the Delphi Component Palette, bands and groups, and the RB Component Palette. It also provides troubleshooting tips and a history of RB versions.

RAP Help

RAP Help is accessible from the RB Help file. It offers an explanation of RAP, a set of tutorials, information on programming with RAP, and a language reference section. It also provides pointers on scaling RAP to meet the needs of your users and extending RAP.

ReportBuilder Newsgroups

The following list of newsgroups are accessible from the Support section of our website:

digital-metaphors.public.reportbuilder.tech-tips
tech tips posted by digital metaphors

digital-metaphors.public.reportbuilder.general
general report building

digital-metaphors.public.reportbuilder.rap
run-time Pascal development environment

digital-metaphors.public.reportbuilder.subreports
using free-form subreports

digital-metaphors.public.reportbuilder.end-user
developing end-user reporting solutions

digital-metaphors.public.reportbuilder.datapipelines
report data access: database, textfiles, stringlists...

digital-metaphors.public.reportbuilder.devices
report output: printer, screen, file, archive...

ReportBuilder Newsgroups - cont.

digital-metaphors.public.reportbuilder.component-writing
custom report components, datapipelines, and devices

digital-metaphors.public.reportbuilder.beta
beta program for upcoming releases

Take advantage of these newsgroups: they afford you the opportunity to peer into and / or join the ReportBuilder community. Please read the set of guidelines before posting.

Further Support

The newsgroups are an excellent resource for technical support, but if you prefer other methods, they are available:

- Mail: support@digital-metaphors.com

Please include a concise description of the issue and specify the following information: Delphi version, ReportBuilder version, Operating System, and Printer model.

- Phone: 972.931.1941

Pay per minute support from a Digital Metaphors engineer is available for US $3.00 per minute. Digital Metaphors engineers are available from 9:00 am to 4:30 pm (Central Time), Monday through Friday. Please note that using e-mail is the preferred method for reporting bugs. Phone support is charged on a US $3.00 flat rate regardless of subject matter.

- Fax: 972.931.7865

INDEX

D

E

Q

R

S